VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Rozšíření serverové části programu Netcon

Extending the Server Program NetCon

2011 Jan Bajer

Zadání

Rozšiřte a upravte bakalářskou práci „Vzdálené zapínání a vypínání PC“ v následujících oblastech:

1. Implementujte možnost konfigurace serveru pomocí konfiguračního souboru.
2. Přidejte volbu pro použití běžné komunikace nebo zabezpečené komunikace prostřednictvím SSL.
3. Systém koncipujte pro použití protokolu IPV6.
4. Vytvořte server pro platformu Linux i Windows.
5. Doplňte server o možnost vzdáleného zapínání PC pomocí Wake on LAN.
6. Konzolovou ovládací aplikaci upravte, tak aby umožňovala komunikaci pomocí SSL i nezabezpečenou komunikaci.
7. Ovládací aplikaci koncipujte tak, aby šla jednoduše použít ve skriptech jazyka Bash, Perl apod.

V úvodních kapitolách práce popište architekturu programu Netcon a komunikační protokol. Výstupem všech částí práce bude instalační balíček pro danou platformu. Postup instalace dobře popište.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum: Podpis: _____

Jan Bajer

Poděkování

Tímto chci poděkovat svému vedoucímu Ing. Davidu Seidlovi za vedení a inspiraci k mnohým vylepšením při tvorbě této aplikace a i celé mojí diplomové práce.

Vedoucí práce: Ing. David Seidl

Abstrakt

Cílem této práce je rozšíření serverové a konzolové ovládací části systému Netcon, jež byla realizována v bakalářské práci „Systém pro vzdálené zapínání a vypínání PC na učebnách“. Jde o rozšíření, která významným způsobem zlepšují možnosti nasazení serveru na širší škále sítí i operačních systémů. Jedná se například o rozšíření, jež umožňuje používat systém nezávisle na verzi IP protokolu či s nastavením několika různých úrovní zabezpečení komunikace. Během jejich implementace došlo k mnoha změnám v koncepci i návrhu serveru, které byly nutné k plnému a efektivnímu přidání nových funkcí, a které budou dále rozvedeny v textu. V této práci je rovněž popsán postup tvorby knihoven, nezbytných k chodu aplikace a instalačních balíčků pro jednotlivé platformy včetně jejich nasazení.

Abstract

The goal of the diploma thesis is to extend the server and console-based modules of the Netcon system that were discussed in the bachelor thesis titled "System for remote startup and shutdown of computer systems in classrooms". These extensions enable the system to be deployed on a broader scale of networks and operating systems and include such features as a possibility to use the system independent on a version of the IP protocol or to set various communication security levels. There are several major changes in both conceptual level and implementation of the server all of which were required in order to add new functionality in an efficient way. Further information about those changes are included in the following text as well as a procedure of libraries development needed for the application to run properly and a set of installation packages meant for each of the platforms along with their deployment details.

Klíčová slova

C++, bezpečnost, SSL, OpenSSL, knihovna, nasazení, komunikace, počítačová síť

Key words

C++, security, SSL, OpenSSL, library, installation, communication, computer network

Obsah

1. Úvod

1.1. Úvodem

V dnešní době stále se rozšiřujících možností informačních technologií a jejich nasazení do dalších a dalších činností života rostou taktéž nároky na efektivní zjednodušování jejich správy a ovládání. Mezi taková zvyšování efektivity patří i možnost provádět různé úkony na počítači, u kterého se právě fyzicky nenacházíme. Existuje velké množství aplikací, které různým způsobem rozšiřují tyto možnosti, ať už se jedná o ovládání jednoho počítače přes „Vzdálenou plochu“, kdy je na monitoru kompletní plocha vzdáleného PC či pomocí konzole, kdy je daný stroj (a nemusí se v tomto případě jednat přímo o počítač) ovládán pomocí jednotlivých příkazů posílaných přes síť.

Obsahem této práce je pokusit se popsat rozšíření a nasazení určitých částí – v tomto případě serverové a konzolové ovládací části systému Netcon, jež by následně měla umožňovat vzdáleně zapínat, ovládat a zjišťovat stav celé skupiny počítačů tzv. „jedním kliknutím“. Tento systém byl v základu navržen a realizován v bakalářské práci „Systém pro vzdálené zapínání a vypínání PC na učebnách“. V této oblasti vzdálené správy není mnoho hotových řešení, která by umožňovala hromadnou kontrolu nad skupinou PC, případně se jedná o drahá komerční řešení. Při rozšiřování o nové funkce došlo ke značnému přepracování jak serveru, tak i komunikačního protokolu i celého návrhu a tím byla podstatně zvýšena rychlost a efektivita. Z důvodu tak rozsáhlých změn a také již existující ochranné známce na název „Netcon“ bylo přistoupeno i ke změně názvu systému na Binnary Control System. V práci a zdrojových souborech je používána zkratka bcontrols pro server a bcontrol pro ovládací konzolovou aplikaci.

Během popisu fungování serveru a ovládací aplikace bude často použito vyjádření „systém“, které popisuje chování serveru navenek k ostatním komponentám, jako jsou koncoví klienti, vykonávající požadované operace. Tyto komponenty zde nebudou nijak rozebírány, případně jen okrajově při popisu protokolu, pomocí kterého je realizována komunikace.

1.2. Kapitoly

V první kapitole je popsán nový návrh systému a jeho komunikace a srovnán oproti předchozímu návrhu. V této části systému došlo ke značnému přepracování, které bude dále popsáno včetně porovnání jeho výhodnosti.

Druhá kapitola se zabývá realizací systému, čili popsáním použitých vývojových nástrojů a knihoven pro jednotlivé operační systémy, jež jsou nezbyté pro chod serveru a samozřejmě také pro jeho úspěšnou kompilaci. Jsou zde vypsány a rozebrány všechny části serveru a způsob jejich fungování včetně rozdílů mezi koncovými platformami. Druhá a kratší část této kapitoly se zabývá popsáním fungování konzolové ovládací aplikace, u které došlo také k nemalému množství úprav a změn.

V třetí části se nachází postup nasazení, kde je ukázána tvorba instalátorů pro všechny zvolené platformy a jsou rozebrány rozdíly v tvorbě a fungování těchto skriptů mezi operačními systémy.

2. Návrh systému a komunikace

Tato kapitola by měla čtenáře uvést do principu fungování serveru Netcon. Je zde v základu rozebráno i fungování předešlé verze serveru a taktéž původního komunikačního protokolu pro lepší porovnání novinek a změn, kterých bylo v průběhu návrhu a vývoje dosaženo, a které mají nemalý vliv na celkovou funkčnost a efektivitu.

2.1. Původní návrh systému a komunikace

Původní návrh systému Netcon spočíval v serveru, jako v centrálním prvku celého systému. Propojoval ovládací část systému s koncovými klienty. Při vykonávání jednotlivých úloh docházelo k připojení ovládací části aplikace, od které byl obdržen požadavek, jenž byl serverem zpracován, a následně docházelo k jeho předávání zvoleným klientům na cílových počítačích. Předání bylo realizováno tak, že se server přímo připojoval na koncové klienty, kterým odeslal požadovanou operaci. Pro každý cílový počítač bylo vytvořeno na serveru nové klientské vlákno. Po vykonání úlohy došlo k předání zprávy o dané operaci zpět serveru, po kterém nastalo ukončení vzájemné komunikace s klienty a tím i ukončení vytvořených klientských vláken. Server informaci od klientů zpracoval a celkový výsledek vrátil ovládací části systému. Ta provedla rozložení a vypsání celkového výsledku, po němž došlo k odpojení od serveru. Všechny ukázkové části systému fungovaly pouze jako konzolové aplikace, jejich použití jako linuxových daemonů či služeb bylo popsáno jen v rámci návrhu.

Původní komunikační protokol jednotlivých komponent byl velmi jednoduchý a spočíval v požadavku, který se skládal z názvu operace a seznamu IP adres, které měly tento úkol provést. Veškerá komunikace probíhala zabezpečenou cestou a to pomoci SSL, která byla implementována pomocí knihoven OpenSSL. Jako příklad požadavku můžeme uvést:

info IP IP IP IP

Při obdržení tohoto požadavku ho server rozdělil podle jednotlivých IP adres a předal jim již jen název vyžadované operace – v tomto případě *„info“*. Během jejího provádění zůstalo spojení i vlákna na serveru stále aktivní, dokud nebyla obdržena od všech klientů odpověď. Odpověď mohla mít například tento tvar:

IP odpověď1 IP odpověď2

Podporované operace komunikačního protokolu:

- vypnout
- restartovat

- info
- pokracuj

Operace s názvem *„pokracuj“* informovala o úspěšném navázání komunikace mezi ovládací aplikací a serverem, po jejímž obdržení pokračovala komunikace odesláním již konkrétního požadavku. Při doručení operace „info“ server obdržel od koncových klientů odpověď s jejich současným stavem. U ostatních požadavků, jak již jejich název napovídá, mělo dojít k restartování či vypnutí koncového počítače.

Původně realizovaný ukázkový server ani ovládací aplikace neměly implementovánu žádnou možnost konfigurace. Byla v nich pouze vytvořena třída s proměnnými, umožňující snadné doplnění této funkčnosti. Tyto proměnné představovaly vlastnosti, které by bylo možno v budoucnu konfigurovat, jako například název a cesta k certifikátům, komunikační port atd.

Veškeré další informace o původním návrhu celého systému jsou obsaženy v bakalářské práci „Systém pro vzdálené zapínání a vypínání PC na učebnách“.

2.2. Nový návrh systému

V současném návrhu systému je opět centrálním prvkem server, který obsluhuje jak požadavky uživatelů, tak i klientů. Řada primárních prvků z původního návrhu byla zachována, ovšem byla přepracována jejich funkčnost. V základě se jedná opět o modulární systém, kdy je server schopen spolupracovat v podstatě s libovolným typem ovládacího rozhraní či koncovými klienty na různých platformách, jež splňují požadavky na nový komunikační protokol.

Klient - První zde realizovanou částí je ovládací aplikace. Ta je vytvořena jako konzolový program, která by měl umožňovat kompletní a pohodlné ovládání celého systému. Kvůli možnosti použití této aplikace v různých skriptovacích jazycích jako je Bash či Perl se požadované operace volí pomocí parametrů aplikace. Díky tomu by mělo být možné dosáhnout širších možností použití a nasazení. Může se jednat například o automatizované provádění některých úloh, které je potřebné provádět opakovaně či v případě nějaké externí události. V této situaci již nebude potřeba ručního zásahu správce sítě nebo administrátora. Aplikace podle vložených parametrů složí požadavek pro klienty, který odešle serveru. Obdržený výsledek operace zpracuje a vypíše na obrazovku. Při použití ve skriptech je také důležitou vlastností možnost potlačit textový výstup aplikace. Program je snadno přenositelný na různé počítače díky automatizovanému instalátoru. Rozdílem oproti původnímu návrhu je, že tato aplikace je plnou funkční součástí systému, kdežto u původního systému bylo pouze popsáno její fungování. Ukázkový konzolový program sloužil pouze pro demonstraci, jak by takové ovládání mohlo fungovat, a nebyl postaven pro praktické nasazení.

Server - Druhou částí, jež je zde vytvořena, je serverová aplikace, jejímž úkolem je zpracovávat obdržené požadavky od ovládací části a případně předávat tyto úkoly dále ke

klientům na koncových počítačích. Předpokladem pro jeho úspěšné nasazení je fyzický server či jiný trvale běžící počítač, dostupný ze všech klientských stanic i ovládací aplikace. Zde dochází k větším změnám v návrhu hlavně ve vnitřním fungování serveru, ale také v komunikační části. Ta bude rozebrána níže. Díky možnosti nastavení, která je obsažena v konfiguračním souboru, dochází ke značnému rozšíření možností – lze využívat vlastních certifikátů, volit porty ke komunikaci a mnoho dalších důležitých prvků. Tato funkce umožní, aby byl server lépe přizpůsobitelný různým druhům klientů, ať už se jedná ovládací aplikace (pod kterými si lze představit i webová rozhraní) nebo koncové klienty.

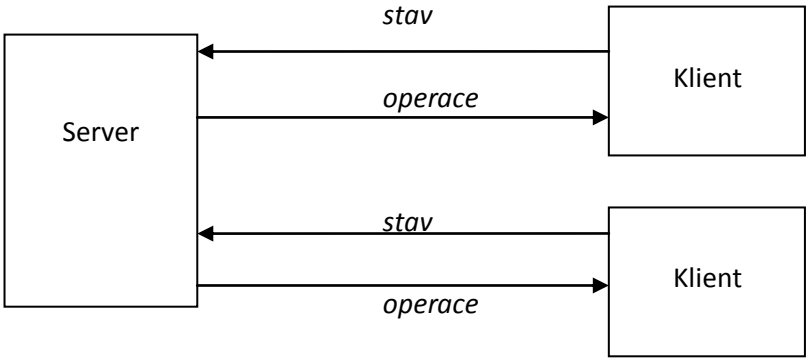
2.3. Návrh komunikace

Komunikace, jako stěžejní prvek celého systému v rámci vylepšení podporuje nové vlastnosti. Samozřejmostí, jako u předchozí verze serverové aplikace, je podpora šifrovaného přenosu zpráv a to směrem ke všem částem systému. Novinkou ovšem je možnost vypnout použití šifrování za pomoci SSL a zvolit komunikaci bez využití této knihovny. Pro základní bezpečnost komunikace je ovšem použito alespoň jednoduché šifrování, které je realizováno vnitřním algoritmem. Tento algoritmus je v případě vypnutí šifrování SSL vyžadován a budou jej muset podporovat všechny zbylé části systému. Veškeré požadavky a odpovědi jsou opět předávány pomocí zpráv. Důležitou vlastností pro budoucí využití je rovněž podpora protokolu IP verze 6, u kterého se očekává velký rozmach v jeho nasazení. Server by z tohoto hlediska měl podporovat v ideálním případě oba protokoly zároveň, aniž by bylo nutné jakkoli řešit jejich volbu či přepínání pomocí nastavení.

Obrázek 1 – komunikace směrem k uživateli

Serverová aplikace řeší komunikaci odděleně pro ovládací část a pro klientskou část systému. V případě ovládací aplikace server naslouchá na určitém portu, zda se nepokouší připojit uživatel s určitým požadavkem, který je předávám serveru. Tato část zůstala návrhově stejná, viz *Obrázek 1*. Ovšem velký rozdíl nastává v komunikaci s klientskými částmi, kdy se již server nepřipojuje na koncové klienty, aby jim předal požadovanou operaci, ale ti se připojí na server. Připojování koncových strojů probíhá periodicky v předem nastavených intervalech. Při každém novém úspěšném připojení je serveru předána zpráva o stavu daného

počítače a je také převzat případný požadavek, který by měl být vykonán spolu s časem, za který se má klient opět připojit. Bližší popis viz *Obrázek 2*. Díky této změně v postupu navazování spojení bude možné nasadit systém i na sítě s překladem adres pomocí NAT. To dříve nebylo možné.



Obrázek 2 – komunikace směrem ke koncovým klientům

2.4. Komunikační protokol

Komunikační protokol je nedílnou součástí každého systému, který využívá počítačovou síť pro přenos svých zpráv či dat. Server i ovládací program nyní obsahují nový a vylepšený protokol, který i přes jen menší úpravy přináší poměrně velké zlepšení ve zpracování samotných zpráv jednotlivými aplikacemi. Také je podporováno velké množství dříve nepodporovaných příkazů, které jsou spojeny s přidáním nových funkcí. Největší změnou oproti původnímu protokolu je využití nového oddělovače mezi parametry příkazu. Jedná se o využití tzv. „pípy“, jež je reprezentována znakem „|“. Požadavek na informace o stavu koncových PC vypadá v nové verzi protokolu takto:

info|ip|ip

Odpověď na takový požadavek má poté tvar:

ip|odpověď|ip|odpověď

Podporované operace nového komunikačního protokolu:

- **info** – zjišťuje informace o koncovém počítači. Server vyžaduje u uživatelské části jako parametr minimálně jednu nebo více adres či doménových jmen počítačů, o kterých má být tento stav zjištěn. Jako odpověď na tento požadavek vrací serverová aplikace adresu a stav PC, který se pod ní skrývá.

- **vypnout** – u uživatelské části je vyžadován opět parametr v podobě adresy počítačů. Server na tento požadavek ihned vrací odpověď, že byla přijata a postupně jej předává dále ke klientské části. K tomuto požadavku je přidán časový údaj, za který server očekává opětovné připojení klienta a ohlášení jeho stavu.

- **vynutitVypnout** – z pohledu serveru i ovládací aplikace se jedná o shodný typ požadavku jako u operace „vypnout“.

- **restartovat** – serverová aplikace očekává jako parametr IP adresy koncových počítačů. Předáván dále ke klientům je opět ve formě příkazu a k němu připojenému času opětovného připojení. Uživateli je okamžitě odeslána zpráva o přijetí nebo o nastalé chybě.

- **vynutitRestartovat** – shodný požadavek na zpracování jako u operace „restartovat“.

- **ping** – parametrem směrem od uživatelské části jsou adresy cílových počítačů. Tento příkaz není od serveru nikam dále předáván, a je jím přímo zpracováván. Jako odpověď na tento požadavek je odesílán seznam adres a jejich zjištěný stav.

- **zapnout** – tento příkaz, který je rovněž zpracován přímo na serveru vyžaduje jako své parametry fyzické adresy síťových karet na počítačích (tzv. MAC adresy), které mají být vzdáleně probuzeny. K ovládací aplikaci server vrací buď nastalou chybu, nebo hlášení o úspěšném zpracování požadavku.

- **resetop** – speciální operace určená pro server. Pokud je obdržena od ovládací části, všechny nastavené operace pro koncové stanice jsou resetovány na výchozí hodnotu. Nevyžaduje vložení žádného parametru.

- **status** – z pohledu konzolového ovládacího programu se jedná o jednu operaci, ovšem ve skutečnosti je tento příkaz složen ze dvou jiných příkazů.

- **nic** – tento příkaz je odesílán serverem pouze směrem ke koncovým stanicím spolu s jedním parametrem, který reprezentuje čas. Určuje, že uživatelem nebyla pro daný PC nastavena žádná operace.

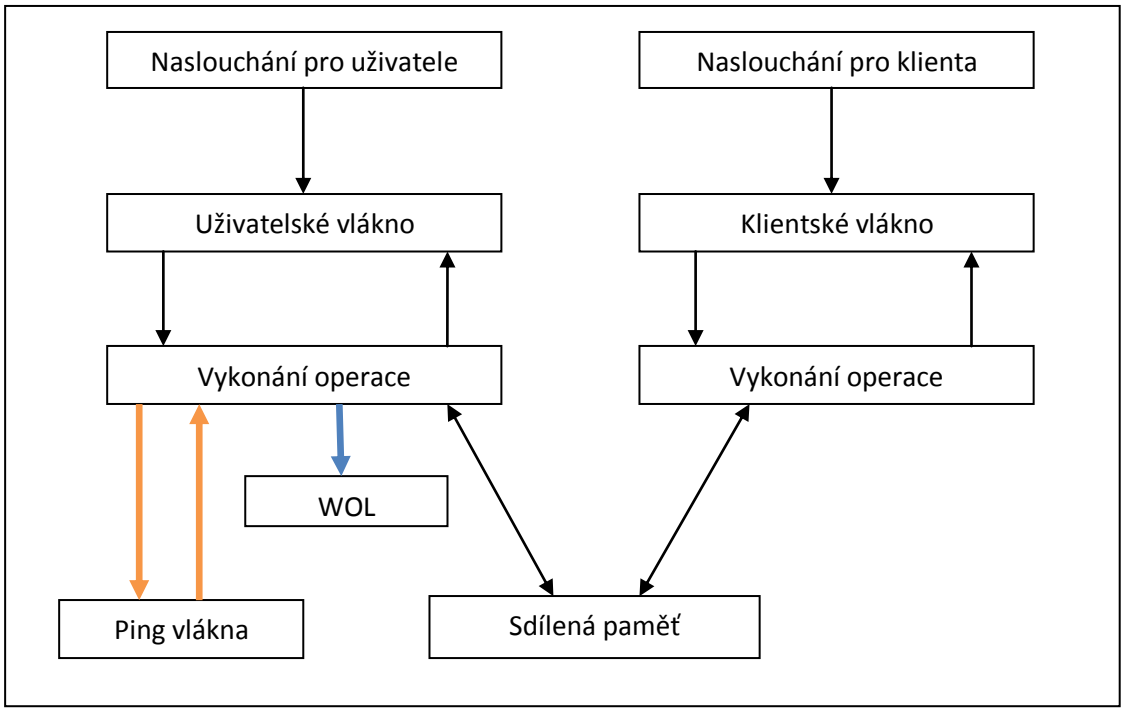
- **verze** – server na tento požadavek vrací číslo své verze.

2.5. Návrh aplikací

2.5.1. Serverová aplikace

Hlavní aplikace, přes kterou prochází veškerá komunikace mezi ostatními částmi systému. Je navržena pro běh jako linuxový daemon i jako služba v operačním systému Windows, avšak je schopna fungovat i jako běžná aplikace s výstupem na obrazovku. V tomto případě je však nutné při jejím spouštění vložit parametry, které zajistí, že se spustí v tomto ladícím režimu. Server obsahuje jako základní prvek dynamickou databázi, ve které jsou obsaženy informace o všech aktivních klientech na koncových PC. Každý klient reprezentuje jednu položku databáze, u které se nachází údaj o stavu daného počítače, jeho adresa, čas posledního připojení k serveru a operace, jež by měl klient vykonat. V případě, že pro PC není k dispozici žádná požadovaná operace, obsahuje tato položka údaj „nic“, který je zároveň i jedním z operací komunikačního protokolu. Tato databáze je vytvořena vždy při spuštění.

Program jako první krok po vytvoření prázdné databáze provede načtení všech konfigurací, po kterém následuje vytvoření dvou hlavních pracovních vláken. Každé z těchto vláken naslouchá na určitém portu. Na jeden z nich dochází k připojování uživatelů s požadavky a ke druhému se připojují klientské aplikace. V případě, že je povoleno použití zabezpečené komunikace pomocí SSL, je při každém připojení provedeno ověření certifikátů.



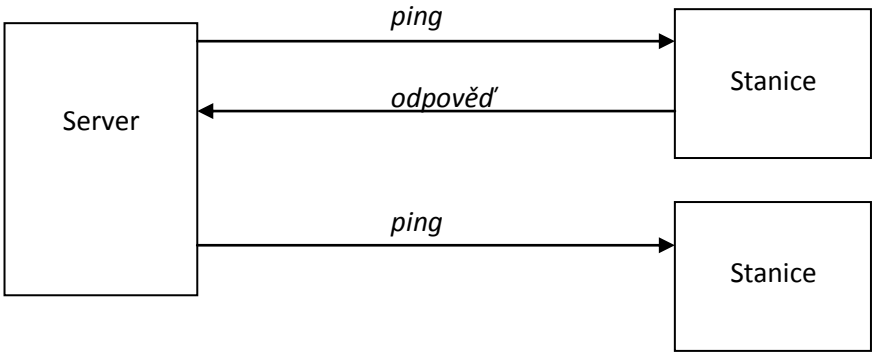
Obrázek 3 – vnitřní návrh serveru

Pokud se připojí nový uživatel s určitým požadavkem k uživatelskému vláknu, naslouchajícímu na zvoleném portu, dojde na serveru k vytvoření uživatelského pracovního vlákna. To se postará o rozložení tohoto požadavku na jednotlivé adresy, jež jsou poté vyhledány v databázi. Pokud je daná adresa úspěšně nalezena, je od ní podle typu operace

buď načten údaj o stavu stanice, nebo je k ní naopak uložena daná operace. V tom případě je jako informace o stanici vráceno, že byl požadavek přijat. V tomto pracovním vlákně je posléze složena odpověď zpět pro ovládací aplikaci. Po jejím odeslání dochází k ukončení komunikace a tím i celého vlákna. Server je schopen v jednu chvíli zpracovávat požadavky více uživatelů, jejich přesný počet lze řídit pomocí konfigurace.

V případě připojení klienta z koncového PC je vytvořeno nové pracovní vlákno z klientského vlákna pro naslouchání, jenž nejprve zjistí adresu připojovaného počítače. Podle ní pak prohledává databázi, ve které se pokouší najít tuto adresu. V případě, že je nalezena, je u ní aktualizovaný stav stanice a případně načten úkol k vykonání. Ten je poté spolu s časovým údajem odeslán zpět klientovi. Po úspěšném doručení je z databáze nastavená operace odstraněna. Pokud adresa v databázi nalezena není, je vytvořena nová položka, do které jsou uloženy obdržené údaje. Zbytek položek, jako je například operace k vykonání, je při vytváření položky vyplněna automatickými údaji. Poté je komunikace ukončena a dochází k ukončení klientského vlákna. Server je opět schopen zvládat připojení většího množství klientů naráz a je možné řídit jejich počet pomocí konfigurace. Pokud nedojde k opětovnému připojení klienta v nastaveném časovém intervalu spolu s pevně danou časovou rezervou, je z databáze odstraněna položka o tomto klientovi a je považován za odpojeného.

Jako jeden z úkolů může být od uživatele obdržen i požadavek „ping“, který by měl zjistit, zda je daný počítač zapnutý a připojení v síti. Dorazí-li tento příkaz, je opět vytvořeno nové vlákno, ale je v něm postupováno již bez použití databáze. V uživatelském vlákně, zpracovávajícím tento požadavek, jsou vytvořena další vlákna, kdy každé zpracovává jednu adresu, u které má být zjištěn stav. Poté, co všechny vlákna dokončí operaci a jsou ukončena, je z výsledků složena zpráva zpět pro uživatele.

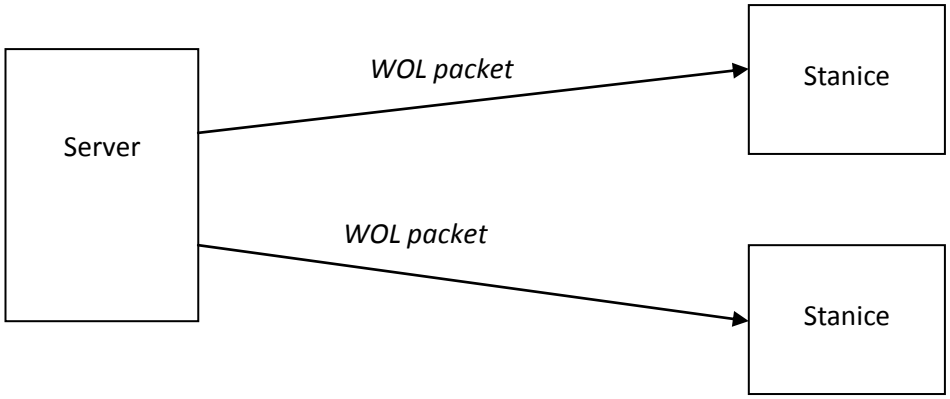


Obrázek 4 – Operace ping

Další variantou, která může na serveru nastat, je obdržení operace „wake“ viz *Obrázek 5*. Tato operace reprezentuje požadavek na zapnutí vzdálených počítačů pomocí funkce Wake on LAN. Uživatelské vlákno v tomto případě postupuje podobně jako u operace „ping“ s tím rozdílem, že každé od něj vytvořené vlákno obsluhuje jednu MAC adresu koncového počítače a tyto vlákna nepracují paralelně, ale po dokončení jednoho je teprve vytvořeno další. Tento postup byl zvolený z důvodu zátěže na elektrickou síť, která by byla nárazovým zapnutím

velkého množství stanic značně zatěžována. Mohlo by tím pádem dojít i k výpadku proudu. Pomocí nastavení lze zvolit i prodlevu mezi spouštěním jednotlivých vláken. MAC adresa je ve vlákně použita k odeslání správného požadavku na zapnutí vzdáleného PC. Server po zpracování všech MAC adres uživateli okamžitě odesílá výsledek, jenž informuje o úspěchu či případně chybě při zpracování. Poté je ukončeno spojení a uživatelské vlákno je ukončeno.

Velký vliv ve schopnosti serveru zpracovávat větší počet uživatelských i klientských vláken hraje výkon samotného hardware, na kterém se serverová aplikace nachází. Je vhodné proto upravit tuto část konfigurace tak, aby odpovídala možnému maximálnímu zatížení daného počítače. Server je navržen tak, aby při překročení maximálního nastaveného počtu klientů či uživatelů došlo k jejich odmítnutí.



Obrázek 5 – Operace wake

Zde popsaný návrh přináší do komunikace mezi serverem a uživatelskou aplikací mnohonásobné zlepšení v rychlosti předávání zpráv. Ta byla u předchozího serveru značně omezena dobou, po kterou se server pokoušel připojit na koncové klienty a poté ještě prodloužena dalším čekáním při zjišťování jejich stavu pomocí ICMP packetu. Proto bylo přistoupeno ke změně v této části a úplnému oddělení zjišťování stavu koncových PC do nové operace s názvem „ping“.

Rozvržení tříd v serveru zůstalo v základu zachováno, ovšem byly upraveny a také bylo přidáno několik nových tříd, jež pomáhají realizovat nově přidané funkce. Celkový počet tříd se v serveru tím pádem zvednul na sedm. Hlavní třídou je **Nastaveni**, která obsahuje veškeré načtené hodnoty z konfiguračního souboru.Od této třídy je dále vytvořena ve dvou instancích třída **Spojeni**, která obstarává naslouchání pro připojení uživatele či klienta. Určení, která instance bude naslouchat pro uživatele a která pro klienta včetně nastavení je dána konfigurací. Logikou jsou obě instance shodné.

Od této třídy je dále podle typu její instance vytvářena nová instance buď třídy **Uzivatel** nebo **Klient**, které mají na starost samotnou komunikaci s uživateli a klienty. Tyto třídy dědí od třídy **Spojeni** a využívají díky tomu určité její metody, které jsou shodné pro obě třídy. Třída **Klient** dále nepracuje s dalšími třídami jako třída **Uzivatel.** Z té mohou být ještě vytvořeny instance tříd **PingPC** a **WakeUP**, o které byl server rozšířen. První z nich obsahuje veškeré proměnné a metody nutné k odeslání a přijetí ICMP packetu, který má za cíl zjistit

stav vzdálené stanice. Druhá m za cíl kompletně obsloužit odeslání tzv. magického paketu, jež by měl umožnit zapnutí vzdálené stanice.

V serveru se nachází ještě jedna speciálně vytvořená třída, která neobsahuje žádnou logiku ani funkčnost. Její název je **Prvek** a reprezentuje pomocí obsažených proměnných právě jeden prvek v databázi. Každá instance této třídy je umístěna v datové struktuře.

Všechny instance všech tříd mají možnost přistupovat k hlavní třídě **Nastaveni** a její instanci, jež je jim předána. Díky tomu mohou všechny třídy přečíst z k tomu určených metod potřebnou konfiguraci a podle té provést své nastavení.

Důležitou roli hraje i rozvržení a obsah konfiguračního souboru. Ten je rozčleněn na hlavní část a 2 další sekce. V hlavní části je možné nastavit název sekcí, které mají být použity jako klientská a uživatelská. Díky tomu je možné mít libovolný počet sekcí s různým nastavením a přepínat mezi nimi pouhou změnou názvu sekce, která má být načtena. V hlavní části je rovněž nastavován způsob a možnosti vytváření logu. Z důvodu oddělené práce s uživateli a klienty obsahuje i konfigurační soubor právě 2 sekce, kdy jedna nastavuje veškeré možnosti pro uživatele od pracovního portu až po maximální počet zároveň připojených uživatelů. Velmi podobné možnosti obsahuje i klientská sekce. Díky jejich oddělení je možné rovněž zvolit využití šifrované či nešifrované komunikace jen u určité části.

Rozvržení hlavní sekce:

logging

- nastavuje hloubku, do které mají být vytvářeny záznamy do logu. Využívá tří úrovní od zapisování pouze kritických chyb až po krokové výpisy (dostupné pouze jako výpis na obrazovku v ladícím režimu)

file_log

- definuje, zda bude vytvářen log do souboru

system_log

- nastavuje, zda bude použit systémový log pro zápis událostí

log_file_name

- název případného souboru, do kterého má být vytvořen log

user_name

- název sekce, která má být použita pro načtení konfigurace u uživatelského vlákna

client_name

- ekvivalent předchozího, pouze pro klientskou sekci

Rozvržení uživatelské sekce:

port

- určuje port, na kterém má server naslouchat pro nezabezpečené příchozí uživatelské spojení

standard_connection

- určuje, že bude použito vlákno pro nezabezpečenou komunikaci i v případě, že je použito již jiné pro zabezpečenou

ssl_port

- nastavuje port, který bude použit v případě SSL komunikace

max_users

- maximální počet uživatelů, kteří mohou najednou pracovat se serverem

ssl

- povoluje vlákno, naslouchající na zabezpečenou příchozí komunikaci

max_ssl_users

- maximální počet zároveň připojených uživatelů s využitím zabezpečené komunikace

sec_level

- určuje úroveň ověřování. Je možné nastavit 3 úrovně zabezpečení s využitím SSL a to bez certifikátů, ověření serveru či ověření obou komunikujících stran

sec_depth

- hloubka, do které má být ověřován certifikát. Lze nastavit úrovně 1 – 9 s tím, že 9 je nejvyšší možná. Doporučené nastavení je základní

autority, certificate a key

- nastavují soubory s certifikační autoritou a použitými certifikáty, je-li použita zabezpečená komunikace

hosts

- konfigurační soubor, obsahující seznam adres, jež se mohou připojit k uživatelské části. Pokud neobsahuje žádnou adresu nebo tato hodnota není obsažena, mohou se připojit libovolné adresy

Rozvržení klientské sekce:

port

- určuje port, ke kterému se mohou připojovat klienti

wake_port

- nastavuje port, na kterém dojde k odeslání takzvaného „magického packetu“, jež by měl umožnit spuštění vzdáleného PC

wake_timeout

- časová prodleva mezi jednotlivými pokusy o zapnutí vzdálených PC. Zajišťuje ochranu proti případnému přetížení elektrické sítě náhlým nárazovým spuštěním velkého množství počítačů

wake_repeat

- počet pokusů o odeslání packetu stejné stanici (pro případ jeho ztráty či poškození)

timeout

- čas, odesílaný koncovým klientům. Po jeho vypršení by mělo dojít k jejich opětovnému připojení

ping_timeout

- v sekundách uvedený čas, po jaký má server čekat na navrácení ICMP packetu

ping_repeat

- počet ICMP packetů, které mají být odeslány jedné stanici

max_clients

- maximální počet klientů připojených k serveru v jednu dobu

ssl

- nastavuje, zda má být použito při komunikaci zabezpečení s využitím SSL

Ostatní hodnoty této sekce jsou již shodné s uživatelskou sekcí. Jedná se především o nastavení certifikační autority a certifikátů v případě využití šifrované komunikace. Pokud dojde k odstranění některé z hodnot, je buď použito výchozí nastavení, nebo je uživatel informován o problému.

2.5.2. Ovládací konzolová aplikace

Slouží jako rychlý a pokročilý způsob ovládání celého systému. Základem je skládání parametrů této aplikace při spuštění tak, aby byl vytvořen požadovaný příkaz pro server. Skládáním parametrů je myšleno:

netcon –param1 obsah –param2 obsah2

Aplikace vložené parametry zpracuje pomocí parseru a poté z nich skládá požadavek pro server. Základem je vkládání adres cílových počítačů, které je možné realizovat buď přímým ručním vložením jako parametru nebo načtením ze zvoleného souboru či souborů. Po složení požadavku a adres je následně vytvořeno spojení směrem k serveru a je mu daná operace odeslána. Klient poté čeká na obdržení odpovědi, kterou poté zpracuje a vypíše výsledek,

pokud není zvoleno, aby byl výstup potlačen. Po jejich vypsání je aplikace ukončena. V případě, že programu není vložen žádný parametr, je vypsána na obrazovku nápověda, popisující jeho použití a poté je ukončen.

Program podporuje řadu parametrů, pomocí kterých je možné upravovat jeho chod a vytvářet potřebné operace pro server. Nejdůležitějším parametrem je volba operace, která skrývá velkou řadu funkcí. Určuje se vložením parametru „**o**“ a za ní se vkládají níže popsané operace.

info

- vytváří požadavek se stejným názvem

reboot

- taktéž vytváří požadavek, jehož cílem je restartování vzdálených stanic.

rebootf

- požadavek vynuceného restartování

cancel

- ruší na serveru veškeré předešlé operace, které ještě nebyly provedeny

shutdown

- provede operaci vypnutí zvolených PC

shutdownf

- vypne vzdálené PC vynuceně bez varování pro uživatele

ping

- skládá pro server požadavek se stejným názvem

all

- kombinuje operaci info a ping, které jsou zvláště odeslány serveru a poté je zpracován jejich výsledek jako celek

version

- server na tento požadavek vrátí číslo své verze spolu s případnými doplňujícími informace a ty jsou vypsány spolu s verzí ovládací aplikace na obrazovku.

wake

- odešle požadavek na zapnutí zvolených stanic

Mezi další neméně důležité parametry aplikace patří parametr „**f**“, který reprezentuje soubor s adresami, které mají být předány jako parametry požadované operace. O toto načtení se stará vnitřní parser aplikace, který vyžaduje určitý formát vstupního souboru. Tento formát je popsán v ukázkovém vstupním souboru. Tento vstupní parametr je povinný, až na případ, kdy je vložen jiný s názvem „**m**“. Ten ruší použití parametru pro načtení souboru

a přebírá jednu určitou adresu vzdáleného PC. Lze jej vložit několikanásobně v jednom požadavku.

Ve skriptech využívaným parametrem je „**q**“, reprezentující režim, kdy nejsou na obrazovku vypisovány žádné informace. S tímto parametrem je spojený i jiný, jež zajišťuje případné zapsání výsledků do textového souboru. Je jím parametr „**l**“. Lze jej použít i samostatně a dosáhnout tak zapsání výsledku do souboru spolu s jeho výpisem na obrazovku.

Samostatným parametrem je „**c**“, který zajišťuje opakované odeslání operace a vypsání výsledků v určitém časovém intervalu. Je vhodný, pokud je nutné zjistit například úspěšné dokončení všech operací a jejich výsledků. Dalším samostatným parametrem je „**d**“, sloužící jako zapnutí ladícího režimu. Tento režim reprezentuje podrobné výpisy kroků programu. Je vhodný zvláště v případě odhalování chyb či pádů programu. V případě, že není vložen žádný parametr, je vypsána nápověda s popisem ovládání aplikace.

Třídy jsou v ovládací aplikaci rozvrženy v základě podobným způsobem jako v serveru, jelikož z něj vychází. Původní ukázková ovládací aplikace nepoužívala třídy a jednalo se pouze o jednoduchý zdrojový kód v jazyce C. Základ opět tvoří třída, obsahující veškerou načtenou konfiguraci s názvem **Settings**. Ta je využívána v druhé třídě k nastavení všech hodnot, nutných k práci i komunikaci. Tato třída se nazývá **Connection**. Celá aplikace využívá pouze těchto dvou tříd.

3. Realizace systému

Cílem této kapitoly je v první řadě ukázat, jaké byly pro vývoj použity vývojové nástroje a popsat jejich správnou konfiguraci pro úspěšnou kompilaci. Nedílnou součástí serveru i konzolové aplikace je také řada nových knihoven, doplňujících jejich funkčnost. U nich bude ukázána správná instalace do systému a použití ve vývojovém prostředí, aby bylo snadno možno kompilovat přiložené zdrojové soubory.

Druhou částí této kapitoly je samotný detailní rozbor a popis fungování serveru (a také ovládací konzolové aplikace) na úrovni zdrojových souborů.

Volba programovacího jazyka pro vývoj ovládací konzolové aplikace i serveru zůstává z důvodu pokračování a rozvoje aplikací zachována. Tímto jazykem je C++ s ohledem na jeho použití napříč platformami bez nutnosti velkých změn zdrojových souborů při případném přenosu aplikace na jinou platformu.

3.1. Použité prostředky k vývoji

Linux

Pod operačním systémem Linux je pro vývoj použit operační systému Ubuntu verze 8.04 a 10.04 spolu s běžným textovým editorem s názvem Gedit, jež má podporu zvýrazňování příkazů jazyka C++ spolu s rozšířeným kompilátorem g++. Kompilace probíhá za pomoci souboru **makefile**, ve kterém jsou obsaženy v podobě skriptu všechny nezbytné příkazy a parametry.

Windows

V operačním systému Windows již bylo použito komplexní vývojové prostředí Microsoft Visual Studio ve verzi 2008. Výsledné aplikace běží v tzv. Managed prostředí, stejně jako například aplikace vytvoření v C# a využívají pro svůj chod .NET Framework. Díky tomu je možné využívat i vlastnost automatizované správy knihoven, který zjednodušuje vývoj samotné aplikace.

Použité knihovny

Při vývoji bylo použito několik dynamických knihoven, které značným způsobem usnadnily implementaci některých funkcí jako je například konfigurace, které by jinak bylo nutné vytvořit zcela od základů a značně by se tak prodloužil vývoj. Na obou platformách jsou využity stejné knihovny, hlavně díky jejich poměrně jednoduché přenositelnosti. Původně byly všechny určeny pro operační systém Linux.

První knihovnou, jež je použita hned při inicializaci aplikace, je **libConfuse** ve verzi 2.7. Tato knihovna obsahuje vše pro kompletní vytvoření a správu konfiguračních souborů od úplně základních s pár řádky až po komplexní konfigurace s řadou sekcí.

Další v aplikaci často využívanou knihovnou je **OpenSSL**, která je dnes hojně rozšířena v řadě aplikací na všech platformách. Umožňuje jednoduše používat šifrování komunikace mezi serverem a ostatními částmi systému na několika úrovních. Díky ní lze využít pokročilých metod zabezpečení s využitím certifikátů a jejich ověřování. V původní serverové aplikaci byla komunikace možná pouze se zabezpečením pomocí této knihovny. Podrobnější informace o této knihovně a tvorbě vlastních certifikátů jsou uvedeny v bakalářské práci „Systém pro vzdálené zapínání a vypínání PC na učebnách".

Velmi důležitou roli v serveru hraje knihovna **pthread**, pomocí které je do aplikace dodána možnost využívat vlákna. Na této knihovně stojí návrh serveru a v určité míře i ovládací konzolové aplikace, jelikož v hojné míře využívají vláken na řadě úrovní jako je v případě serveru možnost komunikovat s řadou klientů či uživatelů zároveň. Vlákna na platformě Windows jsou implementována pomocí této knihovny pouze v jednom případě díky specifické vlastnosti semaforů, které obsahuje.

Knihovnou s menším využitím, která ovšem obsahuje poměrně důležitou použitou funkčnost je **log4cplus**. Slouží k jednoduchému doplnění aplikace o možnost vytvoření logu do textového souboru, obsahujícího všechny potřebné údaje. V této oblasti existuje celá řada podobných knihoven. Tato byla zvolena pro velmi jednoduchou implementaci do serveru. Na systému Ubuntu je přidána k aplikaci statickým linkováním, na rozdíl od ostatních knihoven, z důvodů její nepřítomnosti v repozitářích tohoto OS (a všech od Debianu odvozených distribucí). Tím je zajištěno snadné použití serveru i bez nutnosti kompilace a instalace této knihovny.

Instalace knihoven

V operačních systémech Linux jsou téměř všechny popsané knihovny snadno dostupné přes správce balíčků, kde je potřeba je nainstalovat. Pro kompilaci zdrojových kódů je nutné nainstalovat nejen základní knihovny, ale i jejich vývojové části, jež obsahují v názvu řetězec „dev". Knihovna **pthread** je v operačním systému Ubuntu dostupná již v základní instalaci i pro vývoj. Pod operačním systémem Windows nejsou tyto knihovny v základu obsaženy a je nutné je doplnit. Je možné použít buď již zkompilované verze, pokud jsou dostupné ke stažení nebo provést vlastní kompilaci. Pro vlastní kompilaci knihoven, u nichž nebyly dostupné již zkompilované verze, byly použity tyto prostředky:

- ActiveState Perl
- Visual Studio 2008 a 2010 (Visual C++)
- Netwide Assemebler (NASM)

Veškeré knihovny byly původně vytvořeny pro Visual Studio 6 nebo Visual Studio .NET což si vyžádalo převedení všech projektů pro vyšší verze, ve kterých byla provedena kompilace.

Podrobnější informace pro jejich případnou tvorbu jsou vždy dostupné v popisu u stažených zdrojových kódů.

Tvorba a instalace certifikátů pro uživatele i server pro zabezpečenou komunikaci je popsána v práci „Systém pro vzdálené zapínání PC na učebnách“. Tyto certifikáty hrají důležitou roli v zabezpečení zpráv v případě, že je zvolena komunikace za pomoci šifrování. Server je schopen pracovat s několika úrovněmi zabezpečení těchto certifikátů, jež je možné upravit pomocí konfigurace.

3.2. Kompilace zdrojových kódů

3.2.1. Linux

Veškeré zdrojové a hlavičkové soubory jsou umístěny v hlavním adresáři aplikace (Může se jednat o server i konzolovou ovládací aplikaci). Je zde umístěn i soubor **makefile**, jež obsahuje skripty nutné ke kompilaci. Výsledný spustitelný soubor je po dokončení kompilace vytvořen vždy v podadresáři s názvem „bin“. V tomto adresáři mohou být umístěny různé soubory použité při testování. Samotná kompilace je prováděna pomocí konzole, která je nastavena na adresář se zdrojovými soubory. Do této konzole je vložen příkaz **make**, který se postará o zpracování skriptů obsažených v **makefile**.

Makefile se skládá z řady příkazů, vedoucích k výsledné aplikaci. Při jejich tvorbě je využito i proměnných. Před samotným začátkem překladu jsou vymazány všechny původní objektové soubory spolu s předchozí spustitelnou aplikací.

Odstranění původních souborů

```
rm -f *.o
rm -f ${program}
```

Kompilace do objektových souborů

```
${CC} ${CFLAGS} ${SRC} -c
```

Parametr „**c“** určuje, že budou zpracovávány zdrojové soubory. Ostatní části příkazu jsou proměnné, jež obsahují samotné příkazy a jejich parametry.

${CC} – obsahuje název zvoleného kompilátoru, v tomto případě g++
${SRC} – seznam použitých zdrojových souborů, předávaných jako parametr pro g++
${CFLAGS} – obsahuje parametry, předávané ke zpracování kompilátorem

- **Wall** zapíná vypisování na případné problémy, jež by mohly nastat
- **Wextra** produkuje další možné varování
- **Ansi** zapíná specifikaci jazyka C podle normy ANSI

- **O2** povoluje optimalizaci, určenou ke zrychlení chodu aplikace
- **march=i686** zadává jako cílovou platformu i686
- **msse** zapíná podporu po kompilaci s využitím instrukcí SSE
- **mmmx** povoluje využití instrukcí mmx při kompilaci

Kompilace objektových souborů

```
${CC} ${CFLAGS} ${OBJ} -o ${program} ${OPT}
```

Provádění tohoto příkazů je podmíněno dokončením překladu do objektových souborů. Opět je použita řada proměnných k usnadnění kompilace. Některé z nich jsou shodné s proměnnými použitými při překladu do objektových souborů. V tomto případě je místo parametru „**c“** použit parametr **„o“**, který zajišťuje slinkování objektových souborů.

${OBJ} – seznam objektových souborů, ze kterých bude složena aplikace
${program} – název souboru aplikace spolu s cestou, kam má být vytvořena
${OPT} – reprezentuje seznam knihoven, které jsou nutné k úspěšnému složení programu

- **lssl** jež zajišťuje použití SSL knihoven při kompilaci
- **lpthread** přidává podporu knihovny **pthread**
- **lconfuse** přidává knihovnu **libConfuse**
- log4cplus je reprezentován statickou knihovnou, která je předána kompileru pomocí přímé cesty **/usr/local/lib/liblog4cplus.a**

Kompilace konzolové ovládací aplikace se liší pouze v několika málo částech. Jedná se o nevyužití knihovny **log4cpp** z důvodů výpisů všech nutných údajů přímo do konzole bez nutnosti vytvářet log o chodu programu. Také nejsou použity parametry pro přidání podpory SSE a MMX a dalších optimalizací, jelikož výkon aplikace je dostatečný i bez jejich využití. Jedinou použitou optimalizací je **O2**, i přes její sporný přínos u výkonu. Rozdíl v časech kompilace při jejím využití i bez ní je téměř nepozorovatelný.

3.2.2. Windows

U operačního systému Windows jsou použity dva projekty, vytvořené ve vývojovém prostředí Visual Studio 2008. Nastavení kompilátorů je shodné jak u serveru, tak i u ovládací aplikace. O kompilaci a správu všech souborů se stará samotné vývojové prostředí. Základním prvkem pro úspěšnou kompilaci je použití **Common Language Runtime Support** v nastavení každého projektu, jež zajistí vytvoření Managed aplikací a umožní tak podstatně jednodušší práci s knihovnami.

Důležité je rovněž správné nastavení cest k potřebným hlavičkovým souborům a knihovnám v konfiguraci samotného vývojového prostředí. Již připravené a kompilované knihovny jsou přiloženy u této práce. Toto nastavení se prování v použité verzi Visual Studia následujícím způsobem.

Properties daného projektu/Configuration Properties/VC++ Derectories

Zde je nutné nastavit položky „Include files“ a také „Library Files“ pro každou knihovnu. Cesty se vkládají postupně za sebe a oddělují středníkem. Popis správných cest je uveden níže.

OpenSSL

Pro „Include Files“ je nutné nastavit adresář:

- **OpenSSL\include**

„Library Files“ je určeno touto cestou:

- **OpenSSL\lib**

U ostatních knihoven je postupováno obdobně.

libConfuse

- **confuse-2.7\src**
- **confuse-2.7\windows\msvs.net\libConfuse\Release**

log4cpp

- **log4cpp\include**
- **log4cpp\msvc7\log4cpp\Release**

pthread

- **pthreads-w32-2-8-0-release\Pre-built.2\include**
- **pthreads-w32-2-8-0-release\Pre-built.2\lib**

Seznam použitých hlavičkových souborů pro server i pro ovládací aplikaci je možné vidět v přiloženém projektu v souboru **stdafx.h**, který rovněž obsahuje vložení knihoven, které mají být použity při linkování. U knihoven, jež nejsou standardní součástí vývojového prostředí, je nutné mít správně nakonfigurovány výše uvedené cesty.

Knihovny nutné pro slinkování:

wsock32.lib

- knihovna, umožňují rozpoznání metod pro práci se sockety

libeay32.lib a ssleay32.lib

- tyto knihovny dovolují linkovat metody používající šifrovanou komunikaci pomocí SSL

advapi32.lib

- nutná pro použití metod, spojených se zápisem do systémového logu

libConfuse.lib

- tato knihovna je použita z důvodů využívání metod pro práci s konfiguračním souborem

pthreadVSE2.lib

- dovoluje rozpoznat metody, pracují s vlákny či semafory z knihovny pthread

log4cplus.lib

- jak již název napovídá, dovoluje používat metody spojené s vytvářením textového logu

Ws2_32.lib

- využívají ji metody spojené se zpracováním adres a doménových jmen

3.3. Implementace serveru

V této části budou nejdříve popsány všechny zdrojové soubory pro snadné dohledání veškerých použitých metod, které budou rozepsány v průběhu podrobného popisu fungování serveru a v něm použitých algoritmů. Rovněž budou zvýrazněny rozdíly mezi implementaci serveru pod operačním systémem Linux a Windows. Jedná se zvláště o rozdíl ve spouštění mezi démonem a službou a řadu dalších menších rozdílů, jako je například inicializace socketu.

3.3.1. Zdrojové soubory

Použité hlavičkové soubory

classes.h

- Obsahuje definice tříd, jejich proměnných a metod

internal_secure.h

- Nachází se v něm definovaná logika po vnitřní šifru systému

includes.h

- Vkládají se do něj ostatní hlavičkové soubory, hlavně z použitých knihoven a systému (standardních knihoven, instalovaných spolu s překladačem g++). Tento soubor je vkládán pouze u systému Linux, u Windows je použitý jiný hlavičkový soubor.

U serveru pro systém Windows jsou z důvodů konverze ze systému Linux navíc ještě další hlavičkové soubory, definující funkce či metody, které nejsou součástí tohoto systému.

getopt.h

- Obsahuje definice metod a proměnných, nutných pro zachování kompatibility při zpracování parametrů aplikace

icmp.h

- Jsou v něm definice kódů a struktur, využívaných při tvorbě a zpracování ICMP packetů

iphdr.h

- Definice IP hlavičky

stdafx.h

- Je vytvořen automaticky spolu s novým projektem ve Visual Studio 2008. Jeho využití je shodné se souborem includes.h pod systémem Linux. Obsahuje navíc vložené knihovny s příponou .lib, které jsou nutné pro linker.

<u>Použité zdrojové soubory</u>

main.cpp (respektive bcontrolsWinService.cpp)

- Obsahuje metody **main()**, **mainmethod()** a **ComBridge(),** které obstarávají inicializaci aplikace a vytvoření vláken, nutných pro naslouchání

check_hosts.cpp

- Nachází se zde stejnojmenná metoda, ověřující zda je IP připojeného uživatele považována za důvěryhodnou

check_mac.cpp

- Implementuje metody **check_mac_read()** a **check_mac_write()**, ověřující fyzickou adresu koncové stanice a upravující je do potřebného tvaru

check_settings.cpp

- Obsahuje část metod, které ověřují dostupnost konfiguračního souboru a ověřují platnost některých načtených hodnot

client.cpp

- Jsou v něm umístěny metody **komunikace()** a **zpracovani()**, zajišťující komunikaci s klientskou částí a umístění či čtení potřebných dat z databáze

data_check.cpp

- Metoda **check_operation()**, jež je v tomto souboru umístěna ověřuje, zda je obdržený požadavek serverem podporován

data_transfer.cpp

- Nachází se v něm metody pro přímé odeslání a přijetí dat a to jak s použitím šifrování, tak i bez něj. Jedná se o metody **prijmi(), odesli(), prijmi_ssl(), odesli_ssl()**

error.cpp

- Stejnojmenná metoda obsluhuje zpracování chyb a informačních zpráv a podle zvoleného nastavení dochází k jejich zapsání do textového či systémového log souboru nebo pouze na obrazovku

itoa.cpp

- Převzatý „as-is“ kód, implementující převod číselné hodnoty na typ „string“.

listen.cpp

- Základem je metoda **navaz_spojeni()**, která podle zvolené konfigurace naslouchá na daném portu a při přijetí spojení vytvoří nové vlákno, které požadavek zpracuje

load_hosts.cpp

- Obsahuje metodu se stejným názvem, která načítá a zpracovává volitelný doplňkový konfigurační soubor, jež obsahuje adresy počítačů, které mají oprávnění se připojit z k uživatelskému vláknu serveru

memory_maintanance.cpp

- Metoda maintanance(), která se zde nachází, obstarává mazání neplatných prvků z databáze. Tyto prvky jsou stanice, kterým již vypršela lhůta spolu s rezervou na opětovné připojení k serveru

ping.cpp

- V tomto zdrojovém souboru se nachází stejnojmenná metoda, zodpovědná za ověření dostupnosti vzdáleného počítače za pomoci ICMP packetu. Pracuje pouze s IPv4 protokolem

ping6.cpp

- Funguje stejným způsobem jako předchozí zdrojový soubor s metodou, avšak již s protokolem ve verzi IPv6

ping_prepare.cpp

- Základem je opět metoda se stejným názvem, jako má zdrojový soubor. Jejím úkolem je rozpoznat verzi IP protokolu, pod kterým má být odeslán ICMP packet k PC v síti. Také spravuje jednotlivá vlákna odesílající tento požadavek a sdílený buffer pro výslednou odpověď

return_settings.cpp

- Nalézá se zde velké množství metod, které jsou definované ve třídě **Nastaveni**. Jejich jediným úkolem je vracet hodnotu obsaženou v konkrétních proměnných, podle kterých je provedeno nastavené aplikace

settings.cpp

- Nejdůležitější obsaženou metodou je **nacti_config()**, jež obstarává zpracování samotného konfiguračního souboru pomocí výše popsané knihovny. Také se zde nachází metoda **restore()**, která v případě nenalezení konfiguračního souboru provádí vytvoření nového se základními hodnotami

settings_init.cpp

- Jedinou obsaženou metodou je **init_semaphore()**, obstarávající správnou inicializaci semaforu

ssl_init.cpp

- Obsaženou metodou je **inicializace()**. Jejím úkolem je provést inicializaci základních ssl prvků, které jsou použity při dalším vytváření zabezpečené komunikace

ssl_prepare.cpp

- Stejnojmenná metoda v tomto souboru využívá prvky inicializované v metodě **inicializace()** k navázání šifrovaného spojení a případnému ověření druhé strany komunikace

user.cpp

- Obsahuje metodu **komunikace()**, náležící třídě **Uzivatel**. Ta obstarává veškerou potřebnou komunikaci s uživatelskou stranou

user_database.cpp

- Zde obsažená metoda **trideni()** obstarává veškerou potřebnou logiku pro práci s uživatelským požadavkem jako je rozpoznání požadované operace a její zpracování

wake_pc.cpp

- Nachází se zde metoda awake(), sloužící k odeslání tzv. „magického packetu“ pomocí funkce Wake on LAN, který by měl zajistit spuštění vzdáleného PC

U serveru jsou navíc obsaženy mimo hlavičkových souborů i některé další zdrojové, které doplňují chybějící či z důvodu kompatibility přenesenou funkčnost z operačního systému Linux.

getopt.cpp

- Sada metod implementujících možnost zpracovávat parametry při spuštění aplikace. Jedná se o převzatou Opensource implementaci, jež je plně kompatibilní se zpracováním těchto parametrů s operačním systémem Linux, z něhož pochází.

gettimeofday.cpp

- Obsažená metoda se stejným názvem obstarává převod formátu času na potřebný formát v sekundách a mikrosekundách, který je nutný z důvodu kompatibility pro metody pracující s ICMP packety

Vnitřní šifrovací algoritmus, definovaný v hlavičkovém souboru **internal_secure.h** jak u všech verzí serveru, tak i ovládacích aplikací, je vytvořený za účelem zabránění odesílání požadavků a odpovědí jako čistého textu v případě, že je zvolena komunikace bez využití SSL. Má za cíl zabránit snadnému rozpoznání zpráv, které si jednotlivé části posílají a jejich případnému podvržení. Pro komplexní zabezpečení je doporučené použít šifrování za pomocí SSL a ověření jednotlivých stran komunikace. Funguje na principu bitového posunu pomocí ASCI hodnoty daného znaku. Každý znak je posunut o ASCI hodnotu znaku šifry, která je v tomto případě pevně stanovená v aplikaci. Při zpětném dekódování je použit postup opačný a to odečtení ASCI hodnoty přijatého znaku o ASCI hodnotu znaku šifry. Změna této šifry není podporována, jelikož tento typ komunikace nemá sloužit jako bezpečný.

3.3.2. Implementace

Cílem této podkapitoly je popsat krok po kroku operace, které server provádí od svého spuštění až po zpracovávání jednotlivých požadavků. Popisuje jak samotné použité postupy a metody, tak i rozdíly v nich mezi jednotlivými použitými platformami.

Při spuštění serveru je jako první krok provedeno nastavení dvou základních proměnných, které určují, zda bude server spuštěn v ladícím režimu s dodatečnými výpisy. Tyto proměnné jsou nastavovány parametry, jež se zadávají při spuštění aplikace a jsou zpracovávány knihovnou **getopt**. Níže popsané operace jsou umístěny ve zdrojovém souboru **main.cpp** případně **bcontrolsWinService.cpp** (a jejího hlavičkového souboru).

a) U serveru pod systémem Windows jsou následně načtené hodnoty vyhodnoceny pomocí podmínek a dochází buď ke spuštění či instalaci služby do operačního systému nebo ke spuštění serveru v ladícím režimu. Samotná inicializace a spuštění služby je vykonáno pomocí automaticky generovaného kódu samotným Visual Studiem 2008. Do tohoto kódu bylo doplněno několik částí pro úspěšné dokončení této inicializace a to vytvoření nového pracovního vlákna do virtuální metody **OnStart()**. Toto vlákno je vytvořeno

pomocí knihovny Threading, jež je součástí .NET Framework a usnadňuje tvorbu služby. Vlákno je definováno v této metodě pomocí

```
Thread^ workerThread = nullptr;
```

a následně je mu při inicializaci přiřazena startovní metoda **mainService()**

```
workerThread = gcnew Thread( gcnew ThreadStart( this,
&bcontrolsWinService::mainService ) );
```

Tato metoda se nachází ve zdrojovém souboru **bcontrolsWinService.cpp**. Je v ní pouze vytvořen objekt třídy **Nastaveni** a jeho vnitřním proměnným „debug“ a „verbose“ je nastavena základní hodnota na „false“ z důvodů spouštění aplikace jako služby. Na závěr je zavolána metoda **mainmethod()**, ve které jsou definovány nové proměnné, se kterými bude později pracováno.

Dále je u serveru pod tímto operačním systémem nutné registrovat zdroj událostí, pomocí kterého bude probíhat zapisování chyb do systémového logu. To je provedeno v této metodě pomocí volání

```
errorHandle = RegisterEventSource(NULL, "Binarry control server");
```

kdy je proměnná „errorHandle“ definována ve třídě **Nastaveni**. První parametr volané metody značí, že je použit log na lokálním systému a druhý je jméno, které bude v logu uvedeno jako zdroj. V případě, že se nelze zaregistrovat do systémového logu, je server ukončen s vypsáním chybové zprávy.

Další operací, jež je nutné vykonat je inicializace socketů, bez nichž není schopen server pracovat. Prvním krokem v této inicializaci je

```
wVersionRequested = MAKEWORD(2, 2);
```

kdy proměnná „vVersionRequested“ je typu WORD a je definována v právě prováděné metodě. MAKEWORD je makro, které určuje minimální a maximální verzi knihovny se sockety, jež je nutná pro chod aplikace. V tomto případě je požadována verze 2. Dalším krokem při inicializace je

```
returned = WSAStartup(wVersionRequested, &wsaData);
```

jež provede samotnou inicializace. Návratovou hodnotu této metody je integer. Prvním parametrem je požadovaná verze knihovny a dalším je WSADATA, se kterou bude dále pracováno při zjišťování nalezené verze

```
if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) !=
2)
```

Pokud není nalezena požadovaná verze 2, je aplikace ukončena s vypsáním zjištěného stavu.

b) U linuxového serveru dochází k inicializaci objektu **Nastaveni** a v něm umístěných proměnných „debug“ a „verbose“ rovnou z důvodů jiného spouštěcího mechanismu při vytvoření démona. Při spuštění jsou také definovány další použité proměnné. Pokud je po zpracování vložených parametrů aplikaci zjištěno, že má být spuštěna jako démon, je zavolána metoda **daemonize()**. Té jsou předány dva parametry typu string a to adresář, kde má démon běžet a také umístění PID souboru s novým identifikátorem aplikace. Ten je při přechodu aplikace na démona změněn.

V této metodě nejprve otevřen systémový log z důvodů možnosti zaznamenat případnou chybu

```
openlog("Bcon server", NULL, LOG_DAEMON);
```

Parametry je určen název, pod kterým bude do logu zapisováno a také informace o tom, že má být zapisováno do části logu pro démony. Následně je v podmínce ověřeno, zda aplikace již není démonem a podle toho dochází k ukončení této metody či jejímu dalšímu pokračování.

Jako další krok metody jsou nastaveny systémové příkazy, které mají být blokovány. Nejprve je vytvořena prázdná množina

```
sigemptyset (&newSigSet);
```

a do ní jsou pak přiřazeny potřebné příkazy pomocí jako například „SIGCHLD“, který zajistí blokování potomka.

```
sigaddset(&newSigSet, SIGCHLD);
```

Po dokončení těchto operací je nastaven handler

```
newSigAction.sa_handler = signal_handler;
```

ten je umístěný ve stejném zdrojovém souboru jako metoda **main()**. Dále je vytvořena další množina, která bude naopak zachytávat systémové signály, jež mají být aplikací zpracovány

```
sigemptyset(&newSigAction.sa_mask);
```

ty jsou nastaveny pomocí

```
sigaction(SIGHUP, &newSigAction, NULL);
```

a jedná se o signály „SIGHUP“, „SIGTERM“, „SIGINT“, které značí uváznutí, likvidaci či přerušení aplikace. Po nich je již proveden **fork()** aplikace a ověření ID jednotlivých procesů po této operaci. Pokud je návratová hodnota z této operace záporná, je aplikace

ukončena s chybovým kódem. Při návratovém kódu větším než 0, značícím, že se jedná o „rodiče“nově vytvořeného démona, je aplikace ukončena bez chybového kódu.

Po nastavení masky pomocí volání **umask()** a nastavení práv na hodnotu 750 je vytvořeno pro potomka nové session ID, podle kterého s ním může být dále pracováno (například může pomocí něj být ukončen zvolený potomek). Změna SID je provedena zavoláním **setsid()**, který v případě úspěchu vrací nenulovou hodnotu.

Jakmile je dokončena změna SID, jsou uzavřeny v cyklu veškeré deskriptory, jejichž hodnoty jsou získány voláním **getdtablesize()**

```
for (int i = getdtablesize(); i >= 0; --i)
```

Po jejich uzavření je nastaven pracovní adresář démona, který byl převzat z prvního parametru, předaného metodě

```
chdir(rundir.c_str())
```

Dále je vytvořen PID soubor, obsahující identifikátor démona. Tvorba tohoto souboru probíhá voláním

```
pidFilehandle = open(pidfile.c_str(), O_RDWR|O_CREAT, 0600);
```

Vrácen je identifikátor tohoto souboru. Název a cesta k tomuto souboru jsou předány metodě v druhém parametru. Další parametry určují, že je soubor vytvářen pro čtení i zápis a práva na něj. Tímto souborem je také zajištěno, že démon bude vytvořen pouze v jedné instanci, jelikož je procesem uzamčen metodou

```
lockf(pidFilehandle,F_TLOCK,0)
```

Ta přebírá jako své parametry identifikátor PID souboru, inicializovaného v předchozí operaci a parametr, určující, že je požadováno uzamčení souboru. V případě návratu záporné hodnoty je již tento soubor vytvořen a uzamčen jinou instancí serveru a dochází k ukončení aplikace. V případě úspěšného uzamčení souboru je získáno aktuální PID démona

```
sprintf(str,"%d\n",getpid());
```

a to je následně uloženo do vytvořeného souboru

```
write(pidFilehandle, str, strlen(str));
```

Po těchto operacích jsou již jen uzavřeny standardní výstupy „STDIN_FILENO“, „STDOUT_FILENO“ a „STDERR_FILENO“ pomocí volané metody **close()**. Tím je ukončena činnost této metody a dochází k jejímu návratu do hlavní metody **main()**.

Další postup je již shodný pro obě verze serveru. U obou dochází v dalším kroku k nastavení generování náhodných čísel tak, aby se nejednalo o pseudonáhodné generování. To je provedeno za pomoci

```
srand(time(NULL));
```

Po tomto nastavení je provedena metoda, nacházející se ve třídě **Nastaveni** a to **init_semaphore()**, ve které je inicializován semafor (respektive tzv. „mutex“). Tato inicializace je provedena u Linuxu využitím

```
returned = pthread_mutex_init(&log_id, NULL);
```

a u Windows voláním ekvivalentu

```
log_id = CreateMutex(NULL, false, NULL);
```

V případě operačního systému Linux je využita knihovna **pthread** a v ní obsažená metoda **pthread_mutex_init()**, přebývající jako parametr identifikátor daného semaforu. Druhý parametr v tomto případě zajišťuje použití základního nastavení atributů tohoto semaforu. U systému Windows je využita ekvivalentní metoda **CreateMutex()**, jež je standardní součástí systému. Identifikátor semaforu je v jejím případě vracen jako návratová hodnota, jež musí být inicializována (respektive nesmí být „NULL“). Parametry této metody jsou nastaveny na základní hodnoty předáním hodnot „NULL“ a „false“. Pomocí těchto parametrů lze u semaforu nastavit například, zda má být sdílen v celém operačním systému, či mu nastavit jméno. Těchto vlastností není v případě serveru využíváno. Identifikátor semaforu se mezi systémy liší i v datovém typu, kdy je pod systémem Linux použit typ „integer“ a pod Windows typ „handle“. Inicializace této proměnné probíhá zde z důvodu jejího využití v metodě pracující s vypisováním informací na obrazovku a do log souboru. Tato metoda může být zároveň volána z velkého počtu vláken a nesmí u ní dojít k souběhu.

Po dokončení inicializace je metoda, zabývající se tvorbou semaforu opuštěna a dochází k návratu do volající metody. V té je dalším prováděným krokem načtení konfigurací, které je provedeno v metodě **nacti_config()**. Zde je využita knihovna **libConfuse**, která umožňuje jednoduše zpracovávat i komplexní konfigurační soubory.

Prvním provedeným krokem je definice tří základních struktur, které reprezentují hlavní části konfiguračního souboru. První z těchto struktur reprezentuje globální část souboru s nastavením, kde jsou definovány hodnoty shodné pro celou aplikaci. Jedná se například o nastavení způsobu, jakým bude vytvářen log a zda má být vůbec vytvořen. Také je zde nastaveno, která sekce bude brána jako uživatelská a která jako klientská pro načítání dat do dalších použitých struktur. Zbylé dvě struktury jsou ony klientské a uživatelské. Při této definici jsou rovněž zvoleny základní hodnoty do všech proměnných pro případ, že některá z nich nebude vůbec obsažena v konfiguračním souboru.

Následuje inicializace základní proměnné typu „cfg*“ za pomocí vytvořených struktur

```
cfg = cfg_init(opts, 0);
```

a poté nutná validace, zda jsou struktury správně vytvořeny

```
cfg_set_validate_func(cfg, "section_user", &cb_validate_bookmark);
cfg_set_validate_func(cfg, "section_client", &cb_validate_bookmark);
```

Toto volání pro ověření struktur bere jako své parametry základní proměnou, přes kterou je pracováno s konfigurací a název proměnných, kterými jsou určeny dané sekce. Poslední parametr je metoda, která provede samotné zpracování těchto struktur.

Po dokončení těchto úkonů je provedeno samotné načtení a zpracování souboru s nastavením

```
returned = cfg_parse(cfg, descr.c_str());
```

Druhý parametr je cesta, ve které má být konfigurační soubor hledán. Ta je nastavena pomocí proměnné operačního systému `ALLUSERSPROFILE` a k ní doplněnému adresáři shodným s názvem aplikace a názvem samotného konfiguračního souboru. Tímto způsobem je dosaženo oddělení samotné aplikace od souborů s nastavením a případnými doplňkovými daty či údaji. Pokud proběhne samotné zpracování souboru úspěšně a nejsou nalezeny chyby, jsou načteny hodnoty do proměnných ve třídě **Nastaveni** pomocí několika metod

```
cfg_getint()
cfg_getbool()
cfg_getstr()
```

Hodnoty ze sekcí jsou získány pomocí metody, která vytváří novou proměnnou typu „cfg*“, přes kterou je přistoupeno dále dovnitř každé sekce

```
cfg_gettsec()
```

Po dokončení načítání je ukončena práce s použitou knihovnou a jsou uvolněny její prostředky

```
cfg_free(cfg);
```

Nezbytným krokem je rovněž ověření platnosti načtených hodnot a to hlavně jejich rozsahu v případě například použitých portů či úrovní zabezpečení. Tento úkol je proveden ve zbylé části této metody, respektive v metodách **check_security(), check_file()** a **check_ssl_depth()**, jež jsou z ní volány.

Rovněž je počítáno s variantou, kdy není konfigurační soubor nalezen při pokusu o jeho otevření metodou **cfg_parse()**. V tomto případě je využita metoda **restore()**, která provádí obnovení konfiguračního souboru ze zálohy. Ta je umístěna ve stejném adresáři, kde původní konfigurační soubor. V případě vážných problémů s konfigurací je vhodné odstranit poškozený konfigurační soubor a nechat provést jeho obnovu do základního nastavení. Tuto operaci lze provést i ručně, jelikož obsah záložního souboru s názvem „restore“ je shodný s konfiguračním souborem v základním nastavení. Tato metoda provádí obnovení tak, že vytvoří nový konfigurační soubor, který otevře spolu se záložním souborem. Ten je celý načten do paměti a následně zapsán do nově vytvořeného souboru.

Další postup po návratu z metody **nacti_config()** po úspěšném ověření všech proměnných je vytvoření nového semaforu s názvem „globalni_id“, jež bude v dalším průběhu programu zabezpečovat omezení přístupu k databázi připojených klientů pouze na jedno vlákno. Tato databáze je definována hned v dalším kroku a je využívána napříč celým programem

```
vector<Prvek> prvky;
```

Každý připojený klient je reprezentován třídou **Prvek**, obsahující veškeré nezbytné informace o vzdáleném počítači. Všechny instance této třídy jsou dále uchovávány v datové struktuře typu „vector“, která umožňuje poměrné snadné přidávání, odebírání a vyhledávání prvků podle dané situace. V této fázi se ve vektoru nenachází žádné uložené prvky. Následuje vyhodnocení načtených hodnot z konfigurace a podle jejich nastavení případná inicializace knihoven, umožňujících použít zabezpečení s využitím SSL.

Následně dochází k vyhodnocení lokální pomocné proměnné, které reprezentuje použití textového log souboru. Do ní je při načtení konfigurace nastavena získaná hodnota. Do této doby je i přes načtené hodnoty z konfiguračního souboru toto logování vypnuté z důvodu neinicializovaných struktur, nutných k vykonání této operace. Na jejím základě jsou inicializovány struktury nutné k funkčnímu zápisu do logu.

```
SharedAppenderPtr myAppender(new
RollingFileAppender(LOG4CPLUS_C_STR_TO_TSTRING(filelog.c_str())));
…
myLogger = Logger::getInstance("bcontrols");
```

Proměnná „myLogger“ je umístěna ve třídě **Nastaveni** a pomocí ní je později možné snadno provést zapsání zpráv do textového souboru. Tvorba těchto struktur je umístěna v hlavní metodě, aby nebylo nutné je vytvářet opětovně při každém pokusu o zápis do log souboru. Na konci podmínky je v případě nutnosti povoleno do té doby zakázané nastavení, umožňující použití tohoto způsobu logování.

Hned po jejich případné inicializaci probíhá další vyhodnocení načtených nastavení. V tomto případě se jedná o možnost vytvořit pro naslouchání na příchozí spojení od uživatele jak vlákno pro šifrovanou komunikaci, tak i pro nezabezpečenou. Pokud nastane takováto situace, je v samotné podmínce vytvořeno nové vlákno a to za pomocí struktury **parametry_vlakna_main**. Ta obsahuje proměnné, jež mají být předány nově vytvořenému vláknu. Jedná se hlavně o proměnné určující

- port,
- současnou instanci třídy **Nastaveni**,
- databázi realizovanou vektorem,
- semafor „globální_id“,
- zda je využito SSL,
- zda se jedná o typ vlákna pracující s uživatelem nebo klientskou stanicí

Pro předání těchto údajů do nového vlákna je využito u systému Linux volání

```
returned = pthread_create(&listen_thread2, NULL, ComBridge,
&argumenty);
```

z knihovny **pthread**. Prvním vloženým parametrem je identifikátor vlákna, s kterým může být dále pracováno. Jedná se o datový typ „integer“. Druhý parametr značí použití základních nastavení pro atributy vlákna. Třetí v pořadí je statická metoda, kterou bude nově vytvořené vlákno spuštěno. V Tomto případě se jedná o metodu **ComBridge()**. Jako poslední parametr je výše naplněná struktura, která obsahuje předávaná data pro vlákno. Návratová hodnota je typu „integer“ a její hodnota musí být v případě úspěchu nulová. V systému Windows je nové vlákno vytvářeno následovně

```
listen_thread2 = (HANDLE) _beginthreadex(NULL, 0, ComBridge,
&argumenty, 0, NULL);
```

Podobně jako při vytváření semaforu u tohoto systému je i zde vracen identifikátor typu „handle“. Pro vytvoření samotného vlákna je použita metoda **_beginthreadex()**, jež je součástí API samotného OS. Použití parametry jsou v základě shodné s ekvivalentní funkcí na systému Linux. Nevyplněné parametry jsou nastaveny základními hodnotami. Tento způsob spouštění nových vláken byl zvolen z důvodů jednoduchého přenosu serveru z platformy Linux při provedení pouze malého počtu úprav ve zdrojových kódech. Návratová hodnota této metody musí být jiná než NULL.

Ve spouštěcí metodě nového vlákna je provedeno pouze několik kroků. Základním prvkem je vrácení ukazatele na strukturu s daty.

```
struct parametry_vlakna_main *p = (struct parametry_vlakna_main *)
data;
```

Dále vytvoření objektu třídy **Spojeni**, ve kterém se nalézá metoda, jež bude zpracovávat příchozí spojení na základě převzatých nastavení.

```
Spojeni *komunikace = new Spojeni;
```

A nakonec samotné volání zvolené metody a předání všech potřebných parametrů ze struktury.

```
komunikace->navaz_spojeni(p->port, p->ssl, p->user, p->globalni_id,
*p->prvky, *p->settings);
```

Pokud je samotné vlákno později ukončováno a dojde k návratu zpět do této metody, je nejprve smazán objekt.

```
delete komunikace;
```

a po něm je již jen ukončeno samotné vlákno zavoláním API funkce

```
_endthreadex(0);
```

Tímto je zajištěno vytvoření i likvidace pracovního vlákna. V hlavní metodě aplikace je po prvním zde popsaném vytváření vlákna vytvořeno další, tentokrát již ovšem bez využití podmínek. Toto vlákno je vytvořeno vždy. K jeho vytvoření je použita nová instance struktury **parametry_vlakna_main**. Tím jsou mu předány stejné typy proměnných, ovšem s odlišným nastavením, než v případě prvního vlákna. Rovněž je použita shodná spouštěcí metoda nového vlákna, v níž je zavolána i stejná metoda **navaz_spojeni()**, ovšem z nově vytvořeného objektu a s odlišným nastavením vstupních proměnných.

Na závěr hlavního vlákna je vyhodnocena poslední proměnná z nastavení ohledně použití šifrované komunikace uživatele a podle ní zavolána z nově vytvořeného objektu třídy **Spojeni** metoda **navaz_spojeni()**. Tím je dosažen chod maximálně 3 vláken, které využívají shodnou metodu k naslouchání jak na příchozí uživatelské spojení, tak i na klientské.

Samotná metoda, jež je spuštěna ve třech instancích, řídí své chování pomocí šesti převzatých parametrů, popsaných dříve u struktury **parametry_vlakna_main**. Jejím základním úkolem je vytvoření socketů, na nichž server naslouchá na příchozí spojení. Při jeho úspěšném navázání je vytvořeno příslušné pracovní vlákno. Na jejím začátku je provedeno nastavení základních vnitřních proměnných třídy i lokálních proměnných metody na základě převzatých parametrů. Rovněž jsou definovány objekty, se kterými bude později případně pracováno. Po nich je část, starající se o alokaci správného množství proměnných a struktur, nutných k vytváření nových vláken. Celá tato část je uzavřena v bloku „try“ a „catch“, zajišťující správné odchycení možné chyby, která by mohla při alokaci nastat. Samotné maximální množství klientských a uživatelských vláken, podle kterého probíhá alokace, je nastaveno v konfiguračním souboru. Toto nastavené množství nelze překročit bez změny konfigurace a restartu aplikace. V případě neúspěchu při alokaci je vlákno ukončeno s ohlášením dané chyby.

V dalším kroku je provedeno vytvoření nového semaforu „threads_id“, nutného k uzamčení použitého čítače vláken. Ten určuje množství právě spuštěných vláken a pomocí něj je rovněž určováno, zda je možné vytvořit další vlákna. Semafor je nutný z důvodů přístupu spuštěných vláken k této proměnné. Samotná inicializace tohoto semaforu probíhá postupem popsaným výše.

Po jeho úspěšném vytvoření je v dalším kroku opět zpracována skupina podmínek, tentokrát vyhodnocující, zda mají být vytvořeny struktury SSL a zda jim mají být jako parametry předány uživatelské či klientské hodnoty. Pokud je jedna z podmínek vyhodnocena kladně, je zavolána metoda **inicializace()**, vracející strukturu typu „ssl_ctx“.

V těle této metody je nejdříve inicializována tato struktura v serverové verzi 3.

```
ctx = SSL_CTX_new(SSLv3_server_method());
```

Podle nastavené úrovně ověřování je rovněž v této struktuře určeno, zda má být požadován certifikát k ověření.

```
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER |
SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
```

Toto nastavení určuje, že má být ověřována druhá strana za pomocí certifikátů a v případě jeho neobdržení má být ukončeno spojení.

```
SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, NULL);
```

Opakem je nastavení, kdy není požadováno ověření klienta a je ověřován pouze server u klientských a uživatelských částí.

V případě nastavení, vyžadující ověření klientů, je nutné provést načtení certifikační autority, podle které bude provedeno samotné ověření získaných certifikátů. Jejich načtení probíhá ve dvou pokusech. Při prvním z nich je testováno, zda se zvolený certifikát nenachází v části s konfigurací serveru

```
_dupenv_s()
```

Tato funkce navrací reálnou cestu systémové proměnné „ALLUSERSPROFILE“, kam jsou ukládána veškeré data jako nastavení a další. K této cestě je připojena cesta k souboru certifikační autority. Takto složená cesta je předána k načtení

```
ret_msg = SSL_CTX_load_verify_locations(ctx, path.c_str(), 0);
```

V případě, že je navrácen návratový kód menší či roven nule, je proveden druhý pokus o načtení, tentokrát je jako cesta k autoritě předán pouze řetězec z nastavení. Nezdaří-li se ani načtení tohoto souboru, je proveden poslední pokus, kdy jsou načteny základní lokace certifikačních autorit.

```
ret_msg = SSL_CTX_set_default_verify_paths(ctx);
```

V případě jejich nenalezení je celá struktura de-alokována a je vráceno „NULL“ k volající metodě. Důležitým prvkem je rovněž nastavení hloubky kontroly obdrženého certifikátu. Tuto hodnotu není vhodné měnit v případě použití základních přiložených certifikátů.

```
SSL_CTX_set_verify_depth(ctx, dept_of_check + 1);
```

Jakmile je certifikační autorita v případě nutnosti úspěšně načtena, je vyhodnocena další proměnná z nastavení, na jejímž základě jsou načítány certifikáty. Postup při jejich načítání je téměř shodný s načítáním autority a jsou k němu využity tyto metody

```
ret_msg = SSL_CTX_use_certificate_file(ctx, path.c_str(),
SSL_FILETYPE_PEM);

ret_msg = SSL_CTX_use_PrivateKey_file(ctx, path.c_str(),
SSL_FILETYPE_PEM);
```

Na závěr metody je nastaven čas, po kterém dojde k vypršení SSL spojení a jejímu ukončení. Zabraňuje případnému uváznutí některého vlákna při ztrátě komunikace.

```
SSL_CTX_set_timeout(ctx, 60);
```

V úplném závěru je navrácena vytvořená struktura „ssl_ctx“ a aplikace dále pokračuje v metodě **navaz_spojeni()**. V té již dále dochází k vytváření samotného socketu, nutného k navazování spojení. Nejdříve je vytvořena struktura, obsahující informace o komunikujících stranách.

```
sockaddr_in6 rsa;
```

Tato struktura používá adresy typu IPv6, ovšem je rovněž díky pozdějšímu nastavení zpětně kompatibilní s adresami IPv4. Dále je nastavena struktura, do níž jsou uloženy informace o typu spojení, jež má být vytvořeno.

```
memset( &hints, 0, sizeof(hints) );

hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_NUMERICHOST | AI_PASSIVE;
```

První z nastavených hodnot určuje, že použitý protokol není pevně definován. Z toho vyplívá možnost využít jako čistě IPv6, tak i IPv4. Další hodnota udává použití streamu dat. Jako poslední je použita kombinace dvou parametrů, kdy první umožňuje později použité metodě správně získat adresu vzdálené stanice (a to jak z řetězce s adresou tak i z doménového jména) a druhý umožňuje správné použití získané adresy v dané struktuře. Po nastavení této struktury je zavolána metoda, jež je v serveru často využívána a v tomto případě umožňuje zjistit, jak má být nastaven konkrétní socket. Rovněž je důležitá pro převod adresy počítače na čitelný text a zpět do podoby, zpracovatelné API funkcemi operačního systému.

```
ret_msg = getaddrinfo(NULL, portArray.c_str(), &hints, &result);
```

Parametr „NULL“, vložený jako první, určuje, že je zjišťována adresa lokálního počítače. Další je vložena hodnota, obsahující port, na kterém má být naslouchán. Struktura, jež byla nastavena v předchozím kroku, je nyní vložena této funkci jako třetí parametr. Nakonec následuje výstupní struktura, do níž budou uloženy výsledky z této metody. Návratový kód v tomto případě nesmí být roven -1.

Dále následuje cyklus, v němž je podle získaných výsledků inicializován socket na maximální dostupnou verzi IP protokolu. Informace k tomuto nastavení jsou získány ze struktury „result“ a uloženy pro potřeby cyklu do lokální proměnné „ptr“.

```
for(ptr = result; ptr != NULL; ptr = ptr->ai_next)
```

V samotném cyklu je nejdříve vytvořen socket. V případě selhání pokračuje cyklus znovu od svého začátku. Tentokrát jsou ovšem aplikovány další získané hodnoty ze struktury.

```
sock_listen = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);
```

Veškeré potřebné údaje jsou zjišťovány ze struktury „ptr“. Také jsou nastaveny doplňkové vlastnosti socketu.

```
setsockopt(sock_listen, SOL_SOCKET, SO_REUSEADDR, (char *) &reuse,
sizeof (reuse))
```

Zde uvedené nastavení umožňuje vyhnout se chybě, která varuje, že adresa je již využívána. Důležité je rovněž vyhodnocení podmínky, zda vytvořený socket pracuje s protokolem IPv6. Na jejím základě je totiž nutné nastavit tento socket tak, aby byl schopen přijímat i adresy protokolu IPv4. Ty budou v takovém případě převedeny na kompatibilní formát s protokolem IPv6.

```
setsockopt(sock_listen, IPPROTO_IPV6, IPV6_V6ONLY, (char *)&onIpv6,
sizeof(onIpv6))
```

V systému Windows je toto nastavení nutné provést, jelikož je v základu tato vlastnost vypnutá. Naopak v systému Ubuntu je v základu povolena. Z důvodů kompatibility a možného použití na širokém spektru systémů je ovšem u všech verzí serveru provedeno povolení této vlastnosti bez ohledu na lokální nastavení systému. V případě tvorby socketu pouze s podporou protokolu IPv4 je tato podmínka vyhodnocena nepravdivě.

Další v pořadí je trojnásobný pokus o nastavení socketu, kterému je tím pádem přiřazena adresa a port, na kterých bude pracovat.

```
ret_msg = bind(sock_listen, ptr->ai_addr, ptr->ai_addrlen);
```

Parametry jsou samotný ukazatel na socket, ukazatel na strukturu s adresou a portem a nakonec velikost předchozího ukazatele. Pokud se nepovede nastavit socket ani na třetí pokus, je uvolněn a je přeskočeno zpět na začátek cyklu, kde dojde k vytvoření nového socketu s nastavením získaným z další části struktury. V případě jeho úspěšného nastavení dochází k přerušení cyklu za pomocí „break“ a metoda pokračuje dalšími kroky. Tím je odstranění předtím vytvořené struktury, níž byl nastavený socket.

```
freeaddrinfo(result);
```

Nejdůležitější a nekonečný cyklus celé aplikace, ve kterém je naslouchánо a jsou v něm rovněž vytvářena koncová vlákna je „while“. V něm je nejdříve vytvořena fronta pro naslouchání

```
listen(sock_listen, LISTEN_COUNTER);
```

Vstupem jsou pro něj identifikátor socketu a hodnota, určující maximální velikost fronty. Ta je nastavena v serveru pevně a nelze ji měnit. Poté je alokována struktura, do které se budou ukládat informace o klientovi.

```
rsa_size = sizeof(rsa);
```

Samotné potvrzení příchozího spojení pracuje v blokovacím režimu.

```
soket = accept(sock_listen, (sockaddr*) &rsa, (socklen_t*)
&rsa_size);
```

Návratovou hodnotou této metody je identifikátor nového socketu, pomocí kterého probíhá komunikace s druhou stranou. Dalšími parametry je v předchozím kroku nastavená struktura a její velikost. Po ní dochází k uzamknutí dalších kroků pomocí dříve vytvořeného semaforu „threads_id“. K tomuto kroku dochází z důvodů práce s čítačem vláken.

```
WaitForSingleObject(threads_id, INFINITE);

pthread_mutex_lock(&threads_id);
```

První metoda pro uzamknutí dané sekce je částí API systému Windows. Tato funkce je specifická tím, že je schopna uzamknout nejen semafor, vytvořený metodou **CreateMutex()**, ale jakýkoli objekt. Rovněž obsahuje parametr, kterým je možné nastavit čas, po jehož vypršení dojde o odemčení semaforu či objektu. U serveru je zvolen neomezený čas čekání. Ten je v případě potřeby možné nastavit v milisekundách.

U systému Linux je využita metoda z knihovny **pthread**. Ta přebírá pouze jediný parametr a tím je identifikátor semaforu. U tohoto způsobu uzamykání nelze nastavit čas, za který zámek vyprší, pouze s využitím jiných metod z této knihovny.

Dále je vyhodnocena skupina podmínek, které ověřují, zda není překročen limit připojených uživatelů či klientů a podle toho dochází k případnému uzavření spojení a odemknutí semaforu

```
ReleaseMutex(threads_id);

pthread_mutex_unlock(&threads_id);
```

Obě zde uvedené metody přebírají již pouze jeden parametr a tím je identifikátor semaforu, který má být nastaven na stav odemčeno. Po této operaci aplikace pokračuje opět od začátku hlavního cyklu, kde naslouchá na další příchozí spojení. V další části kódu je vyhodnocena podmínka, ověřující, zda vlákno pracuje v klientském či uživatelském režimu. Podle toho jsou provedeny dvě různé větve kódu, které vytvoří nové vlákno pro správné zpracování požadavku. Nejdříve zde bude popsáno zpracování obou větví podmínek a poté samotná vykonávaná vlákna.

V první větvi jsou prováděny úkoly, související s uživatelskou částí. Nejdříve je ověřena podmínka, zda má být adresa právě připojeného PC ověřena proti povoleným adresám, jež se mohou k serveru připojit. V případě, že má být ověřena, je nejprve zjištěna adresa vzdáleného PC

```
getnameinfo((struct sockaddr *)&rsa, sizeof(rsa), namebuf,
sizeof(namebuf), NULL, 0, NI_NUMERICHOST)
```

K tomu použitá funkce **getnameinfo()** je kompatibilní jak s IPv4, tak i IPv6 protokolem. Jakmile je tato adresa úspěšné získána, je ověřena proti seznamu načtených adres či doménových jmen, které mají oprávnění pracovat se serverovou aplikací. Tyto adresy jsou načteny v rámci načítání konfigurace ze souboru, který je nastaven v souboru s nastavením. Jedná se o jednoduchý parser, který je blíže popsán v samotném zdrojovém souboru. Jeho výstupem je řetězec ve formátu, umožňujícím snadné pozdější zpracování.

```
adresa.vsb.cz|192.168.1.2|192.168.1.3
```

Ověření samotných načtených adres, oproti povoleným, probíhá v metodě **check_hosts()**. Zde je na rozložení řetězce adres použit „stringstream“ pro jeho poměrně snadné použití s datovým typem „string“. V této metodě je nejprve zpracována adresa pomocí **getnameinfo() a getaddrinfo()**. Jejich výstupem jsou dvě možné reprezentace dané adresy. Toho je u porovnání využíváno z důvodu možnosti vložit do seznamu povolených adres jak doménové jméno, tak přímo adresu. Po tomto úkonu již následuje cyklus, kdy dochází k samotnému porovnání. V něm je opět využito dvou výše popsaných metod, tentokrát však na načtené adresy z daného souboru. V případě nalezení dané adresy v seznamu metoda vrací „true“.

Zpět v metodě **navaz_spojeni()** je podle obdrženého výsledku buď ukončeno spojení a dochází k návratu na začátek hlavního cyklu nebo aplikace pokračuje vytvořením instance třídy **Uzivatel** a dále naplněním struktury **parametry_vlakna** pro nové uživatelské vlákno. V této struktuře jsou obsaženy tyto informace

- objekt třídy **Spojeni**
- nově vytvořený objekt třídy **Uzivatel**
- identifikátor socketu
- zda je povoleno šifrování pomocí SSL
- objekt třídy **Nastaveni**
- identifikátor semaforu pro přístup k databázi
- databázi s prvky
- inicializovaná struktura ssl_ctx

Po naplnění struktury potřebnými daty je vytvořeno samotné nové vlákno, kde je přebrán a zpracován požadavek od uživatele. Vstupní metodou pro toto vlákno je **premosteni_user()**, kde jsou ze struktury vytaženy veškeré informace a předány dále hlavní uživatelské metodě. Tou je **komunikace()**, ke které je přistoupeno přes objekt, vytvořený v předchozím vlákně ze třídy **Uzivatel** a který byl rovněž předán za pomoci struktury.

V případě tvorby klientského vlákna je postupováno obdobně, pouze s rozdílem, že se zde nenachází možnost blokovat připojování určitých adres. Opět je na začátku vytvořena struktura, která tentokrát obsahuje místo objektu třídy **Uzivatel** objekt třídy **Klient**. V ostatních obsažených prvcích je shodná. Vlákno startuje metodou **premosteni_client()**, kde je zavolána metoda, jež již obsluhuje samotného klienta s názvem **komunikace()**.

Jakmile je dokončena daná větev podmínky, dochází k navýšení čítače běžících vláken a poté již k odemknutí semaforu „threads_id“. Současné vlákno aplikace se po tomto vrací zpět na začátek cyklu a naslouchá na další příchozí spojení.

Uživatelské vlákno

Zpět v uživatelském vlákně je v prvním kroku nastaveno několik lokálních proměnných. Následně je dokončeno nastavení SSL komunikace, u níž byla již dříve inicializována struktura, pokud tak vyžaduje nastavení. K tomu dochází v metodě **ssl_prepare()**, které je předána jako hlavní parametr struktura „ssl_ctx“. Na jejím začátku jsou definovány proměnné typu „ssl“ a „x509“, které budou dále použity při navazování zabezpečeného spojení. Dále je inicializována proměnná „ssl“.

```
ssl = SSL_new(ssl_ctx);
```

Inicializační metoda přebírá jako parametr strukturu, ve které se nachází nastavení, podle něhož bude následně nastaveno i ssl spojení. Takto vytvořenou proměnnou je nutné svázat s vytvořeným socketem, na kterém je již navázána nezabezpečená komunikace, aby bylo možné ji využít i k zabezpečené komunikaci.

```
ret_msg = SSL_set_fd(ssl, soket);
```

Po tomto svázání už následuje samotné navázání SSL spojení.

```
ret_msg = SSL_accept(ssl);
```

Jakmile je úspěšně navázáno, je podle nastavení rozhodnuto, zda je nutné vyžadovat certifikát a jeho ověření. V případě, že ano, je vyžádán certifikát a uložen do na začátku metody definované proměnné „cert“.

```
cert = SSL_get_peer_certificate(ssl);
```

Po jeho získání je nutné jej ověřit do nastavené hloubky a podle výsledku buď spojení ukončit, nebo v něm pokračovat.

```
ret_msg = SSL_get_verify_result(ssl);
```

Tato metoda je využívána jak uživatelskou, tak i klientskou částí a funguje na základě předávaných parametrů, se kterými pracuje. Z tohoto důvodu je nutné klientským a uživatelským vláknům předávat objekt třídy **Spojeni**, ve kterém je tato metoda definována. Navrací proměnnou typu „ssl“, se kterou je dále pracováno.

Zpět v metodě **komunikace()** je tato proměnná přebrána a použita k další komunikaci. V případě selhání jakékoli volané metody je uživatelské vlákno ukončeno s výpisem dané chyby a rovněž je snížen čítač aktivních vláken. V případě, že není při komunikaci využito ssl, je v této metodě vygenerována hodnota v rozmezí 10 až 10 tisíc, která bude použita jako šifra pro vnitřní zabezpečení.

Následuje již vykonání samotné komunikace, ke které jsou použity metody **prijmi_ssl()**, **odesli()** a **prijmi()**. Veškeré tyto metody jsou obsaženy ve třídě **Spojeni** a v případě metod,

přijímajících data, navracejí proměnnou typu „string“. Obsahem všech těchto funkcí, je realizovat samotnou komunikaci a práci s daty. Při nešifrované komunikaci vykonávají bitové posuny pro správné odeslání a přečtení dat. Pro samotný příjem a odeslání je využito několika metod.

```
len = SSL_write(prom_ssl, buff.c_str(), buff.length());

len = SSL_read(prom_ssl, read_buff, KAPACITA);

llen = read(soket, read_buff, sizeof(read_buff));

llen = write(soket, buffer, strlen(buffer));
```

Funkce, jejichž název začíná na „SSL“, realizují zabezpečenou komunikaci, ostatní metody používají zabezpečení pouze na základě bitového posunu. V případě jakéhokoli problému v komunikaci je zaznamenána patřičná chyba a celé komunikační vlákno je ukončeno.

Po obdržení požadovaných operací za pomoci uvedených metod je provedena důležitá uživatelská metoda **trideni()**, která se zabývá samotným ověřením a zpracováním obdrženého požadavku. Ten je v ní nejprve převeden na „stringstream“ a následně probíhá zpracování jednotlivých jeho částí. První v pořadí se nachází samotný název vykonávané operace. Ta je ověřena, zda se jedná pro server o platnou a známou operaci. Posléze jsou již postupně zpracovávány možné obdržené úkoly.

Prvním v řadě je „verze“, který do odpovědi pro uživatele zapíše pouze současnou verzi serverové aplikace. Další možnou operací je „ping“, která ověřuje odesláním icmp packetu, zda je vzdálený počítač spuštěný. Samotná příprava této operace se uskutečňuje v metodě **ping_preprare()**.

V této funkci dochází k rozležení zbytku požadavku, který následoval za názvem úkolu k vykonání. Tato část využívá k blokování kritických sekcí semafor z knihovny **pthread**, použitý v serverové aplikaci pro systému Linux. Je použit z důvodů možnost zablokovat jeden semafor druhým globálním. Tato část funguje na principu, kdy je pro každou adresu vytvořeno nové pracovní vlákno, které odesílá samotný ICMP packet a čeká zpět na odpověď. U nich je použit semafor na čítač vláken. Dále je zde obsažen ještě další semafor, jenž čeká na kompletní dokončení všech vláken, po kterém je teprve možné odeslat odpověď zpět uživateli. V tomto případě je využíváno vlastnosti z knihovny **pthread**, která zajišťuje, že není možné zamknout již uzamčený semafor. Tato vlastnost není při současném použití pod operačním systémem Windows plně funkční. Po odemknutí globálního semaforu dojde k opětovnému pokusu uzamknout lokální semafor pro čítač vláken. Toto uzamknutí je provedeno z důvodů možného úplného nedokončení lokálního vlákna, kdy by mohlo být jeho rodičovské vlákno ukončeno dříve, než lokální, které vykonává odeslání ICMP packetu. Tvorba samotných vláken probíhá obdobným způsobem, jako vytváření uživatelských vláken. V každém novém vlákně existuje rozhodná část, která se stará o rozpoznání, jestli obdržená adresa náleží protokolu IPv4 či IPv6. K tomu je využita API funkce **getaddrinfo()**, ze které je poté zjištěn použitý protokol. Na základě podmínek je dále rozhodnuto, pod jakým protokolem bude odeslán použitý ICMP packet. V implementaci samotného odeslání a přijmu ICMP packetu je použito běžných metod, které jsou popsány ve zdrojových souborech či případně

v dostupných zdrojích, uvedených v závěru. Při ukončení všech vláken je metodě **trideni()** vracena složená odpověď pro uživatele, která je poté odeslána.

Další možnou operací, jež uživatelské vlákno zpracovává, je „zapnout“. Parametry jsou v tomto případě fyzické MAC adresy cílových počítačů. Tyto adresy je nejprve nutné ověřit, zda jsou platné a lze je použít pro vzdálené zapnutí. Ověření platnosti adresy probíhá v metodě **check_mac_read()**. Důležitou částí požadavku je rovněž druhá část, jež následuje za samotným názvem operace. Obsahuje adresu, která reprezentuje multicast či broadcast adresu, na kterou budou odeslány požadavky na zapnutí stanic. Tím je určena podsíť, ve které má dojít k zapnutí. Rovněž je napočítán počet stanic, které mají být spuštěny.

Následuje provedení údržby v databázi připojených klientů. Tato údržba je prováděna pouze v případě, že je vykonávána jiná operace než „zapnout“ a „ping“. Údržbu vykonává metoda **maintanace()**, která je definována ve třídě **Spojeni** a provádí tuto činnost pro uživatelská i klientská vlákna.

V této metodě je nejprve zjištěn čas, který je použit k porovnání času, kdy se naposledy připojili klienti, obsažení v databázi

```
time_t now = time(NULL);
```

Před samotným prohledáním databáze na neplatné stanice je nejprve nutné uzamknout semafor, jenž hlídá přístup k této databázi a vytvořit iterátor, který je použitý u samotného prohledávání. Procházení prvků je realizováno cyklem, který prochází celou databázi od prvního do posledního prvku

```
for ( iter = prvky.begin(); iter != prvky.end(); ++iter )
```

V samotném cyklu dochází k vyhodnocování podmínky, ve které je porovnán čas u každého klienta oproti současnému času s rozdílem, který je nastavený v konfiguračním souboru. Tento rozdíl reprezentuje čas, po kterém se má klient opětovně připojit k serveru a předat mu své údaje. K rozdílu je přičtena ještě rezerva 5 vteřin, která je pevně nastavená v aplikaci. V případě nalezení takového prvku dojde k jeho odstranění z databáze. Tím se předejde předání již neplatných údajů uživateli. Po dokončení cyklu je odemknut zámek pro databázi a metoda je ukončena.

Zpět v metodě **trideni()** již probíhá v cyklu „while“ zpracování samotných adres podle obdržených příkazů. V tomto cyklu se nalézá podmínka s dvěma možnými průchody. První nastává v případě operace „zapnout“. V něm jsou zpracovány a opět ověřeny postupně všechny MAC adresy počítačů. Dále je vytvořen objekt třídy **WakeUP**, přes který bude volána metoda, zajišťující vytvoření socketu a odeslání požadavku. Samotná metoda pro vykonání této činnosti je volána ve vnořeném cyklu, který je opakován podle hodnoty nastavené v konfiguračním souboru. Odeslání je prováděno opakovaně z důvodů možné ztráty packetu. Po odeslání nastaveného počtu packetů k jedné adrese dojde k uspání vlákna na předem nastavený čas. Ten reprezentuje ochrannou dobu, aby nedošlo k nárazovému spuštění všech počítačů, jejichž adresa byla předána serveru. Hodnotu této doby je možné nastavit v konfiguračním souboru. Po uplynutí nastavené doby pokračuje cyklus v dalším průchodu.

Alternativní větví podmínky je ověření platnosti vložené IP adresy funkcí **getaddrinfo()**. Po ověření je podmínka ukončena a vlákno pokračuje získáním přímé adresy z vložených údajů

za pomoci funkce **getnameinfo()**. Z takto získané adresy je vytvořena i její varianta ve formát IPv4 adresy mapované do IPv6 protokolu.

Dále je již provedeno uzamčení semaforem pro práci s databází, je vytvořen iterátor stejně jako v případě údržby databáze a ta je celá procházena v cyklu až do případného nalezení hledaného prvku. Rozhodnutí o jeho nalezení je realizováno podmínkou, která porovnává adresu z databáze s adresou obdrženou v požadavku, adresou zpracovanou funkcí **getnameinfo()** a adresou mapovanou do protokolu IPv6. Jakmile dojde k jejímu nalezení, je k dané adrese buď uložena informace o operaci, jež má vykonat, nebo jsou informace přidány do odpovědi pro uživatele. Cyklus je po tomto vrácen zpět na začátek a pokračuje další adresou. Pokud nedojde k nalezení adresy v databázi, je ke zprávě pro uživatele přidána informace o tom, že daná stanice je odpojená. Na závěr je odemčen semafor pro práci s databází a metoda je ukončena.

Po návratu do metody **komunikace()** je provedeno odeslání odpovědi směrem k uživateli za pomocí metod **odesli()** a **ssl_odesli()**. V případě úspěšného vykonání tohoto úkolu je ukončena komunikace a případně uvolněna proměnna typu „ssl“.

```
SSL_shutdown(ssl_user)
SSL_free(ssl_user);
shutdown(soket, SD_BOTH);
closesocket(soket);
```

V případě systému Linux je provedeno ukončení komunikace obdobným způsobem s jediným rozdílem ve volané metodě pro uvolnění socketu

```
close(soket);
```

Po uvolnění socketů a vyčištění paměti je uzamknut semafor pro čítač vláken pro uživatele a ten je snížen o jedna. Tím je ukončeno i celé uživatelské vlákno.

Klientské vlákno

Podobně jako u uživatelského vlákna je vstupem metoda **komunikace()**, tentokrát ovšem ze třídy **Klient**. Stejně jako na začátku uživatelské metody je nejprve v případě nutnosti vytvořena proměnná typu „ssl“ metodou **ssl_prepare()**, jež je popsána u uživatelské části. Následně je provedena první část komunikace, kde jsou od klienta získány údaje. Ty jsou zpracovány v metodě **zpracovani()**. V cyklu je zde procházena databáze s prvky a je hledána adresa připojené stanice. Pokud je úspěšně nalezena, jsou údaje v databázi aktualizovány a rovněž je složena odpověď pro klienta. Ta obsahuje případný úkol, který má být vykonán a časový údaj, po kterém má dojít k opětovnému připojení k serveru. Tím je metoda ukončena. V případě, že se nepovede najít stanici v databázi, je vytvořen nový objekt třídy **Prvek** a v něm obsažené proměnné jsou nastaveny podle obdržených hodnot. Teprve po tomto úkonu je počítač přidán do databáze

```
prvky.push_back(objekt);
```

Rovněž je složena i patřičná odpověď pro klienta, která v tomto případě neobsahuje žádnou operaci k vykonání z důvodů dřívější neexistence tohoto prvku v databázi. Po dokončení těchto úkolů je metoda ukončena.

Zpět v metodě komunikace() je odpověď odeslána klientovi s využitím metod **odesli()** či **odesli_ssl()** a je s ním ukončena komunikace obdobným způsobem, jako v případě uživatele. Ještě před ukončením samotného klientského vlákna je vygenerováno náhodné číslo na základě počtu připojených klientů

```
i = rand() % obj_spojj.pocet_vlaken + 1;
```

Pokud je takto vygenerované číslo rovné jedničce, je provedena údržba databáze s využitím již popsané metody **maintanance()**. Po jejím provedení teprve dochází k ukončení klientského vlákna.

3.4. Implementace konzolové ovládací aplikace

Ovládací aplikace, jež slouží k odesílání požadavků serveru a zpětnému zpracování odpovědí, obsahuje ve své implementaci podstatně méně rozdílů mezi verzí pro systém Windows a verzí pro OS Linux. Níže je uveden obsah zdrojových souborů a rovněž popis fungování této aplikace. Ten je z důvodů rozsahu a zaměření práce více směrem k serverové aplikaci a také z důvodů použití stejných technik a algoritmů popsán ve větší stručnosti. Detailní informace lze nalézt v komentářích u přiložených zdrojových souborů.

3.4.1. Zdrojové soubory

Hlavičkové soubory

classes.h

- Obsahuje definice tříd a v nich obsažených metod

getopt.h

- Obsahuje definice metod a proměnných, nutných pro zachování kompatibility při zpracování parametrů aplikace

stdafx.h

- Nachází se zde vložené knihovny a hlavičkové soubory, které nejsou součástí aplikace

template.h

- Je zde obsažena logika pro vnitřní šifru při nezabezpečené komunikaci. Ekvivalentní jako internal_secure.h

Zdrojové soubory

main.cpp (bcontrolMain.cpp)

- Vstupní metodou, jež se zde nachází je **main()**, případně **_tmain()** ve verzi pro systém Windows)

check_mac.cpp

- Stejnojmenná metoda ověřuje vložené MAC adresy

clear.cpp

- Obsažené metody mají za úkol provést správné ukončení aplikace v případě přerušení jejího chodu. Jedná se o metody **uklidit()** a **zachyceno()**

close.cpp

- Ukončuje vzájemnou komunikaci v metodě **commnication_close()**

comunnication.cpp

- Metoda se stejným názvem zde podle nastavení volá ostatní metody, nutné k započetí a provedení komunikace

config.cpp

- Nacházejí se zde funkce **restore()** a **load_config()**, jejichž úkolem je práce s konfiguračním souborem

connect.cpp

- Uskutečňuje zde v metodě **connection()** samotné nezabezpečené navázání spojení

data_check.cpp

- Obsahuje funkci **check_action()**, která ověřuje platnost vloženého parametru pro akci

error.cpp

- Stejnojmenná metoda vypisuje veškeré zprávy na obrazovku

getopt.cpp

- Knihovna, která umožňuje zpracování parametrů aplikace

help.cpp

- V případě zadání správného parametru vypíše metoda **print_help()** nápovědu na obrazovku

isip.cpp

- Celý soubor tvoří metody **tokenizer()** a **isip()**, jejichž úkolem je zpracovat vložené adresy a ověřit jejich platnost

itoa.cpp

- Převzatý „as-is“ kód, implementující převod číselné hodnoty na typ „string“

load_file.cpp

- Obsažená metoda **read_data()** je pro chod aplikace podstatná, jelikož načítá a zpracovává veškerá data v podobě adres

mainmethod.cpp

- Funkce se stejným názvem obstarává prvotní zpracování parametru operace a podle něj dále upravuje chod aplikace

output.cpp

- Zde se nacházející metody **result_output()**, **file_output()** a **outputs()** zajišťují kompletní zpracování odpovědi, obdržené od serveru

prepare_data.cpp

- Zajišťuje v případě použití složené operace správné zpracování požadavků k odeslání i přijatých odpovědí a to funkcemi **prepare_data()**, **prepare_output() a output_search()**

return_settings.cpp

- Řada obsažených metod umožňuje získání hodnot, načtených z konfiguračního souboru

SSL_init.cpp

- Metoda **inicializace()** se stará o prvotní inicializaci SSL struktur

SSL_transfer.ccp

- Umožňuji ve funkci se shodným názvem navázat a provést zabezpečenou komunikaci se serverem

thread.cpp

- Obsahuje pomocnou metodu **vlakno()**, která pouze provádí postupné vypisování „stavových“ teček během čekání aplikace na odpověď

unesc_transfer.cpp

- Provádí v metodě **clear_communication()** komunikaci nezabezpečenou cestou

3.4.2. Implementace

Implementace ovládací aplikace využívá velmi podobných algoritmů a technik, jako server. Velké množství zdrojových souborů a kódů je zapříčiněno především z důvodů zajištění funkčnosti velkého množství kombinací vložených parametrů. Kvůli tomu je nutné ověřit řadu podmínek a zajistit zpracování různých metod v různém pořadí.

V aplikaci je využito kromě proměnných ve třídách a místních lokálních proměnných v metodách i několik globálních proměnných. Ty jsou prostupovány zdrojovými soubory s využitím „extern“. K jejich použití bylo přistoupeno z důvodů použití v metodách, které nejsou součástí tříd. Jedná se hlavně o ty funkce, které souvisí s náhlým ukončením programu, například při použití kombinace kláves Ctrl + C. V případě obdržení takového ukončovacího signálu jsou v těchto metodách uvolněny prostředky, jako je socket a SSL struktury. Rovněž jsou takto použity proměnné semaforu pro sdílené proměnné. Jednou z nich je „progress“, která určuje, zda má být vlákno ukončeno nebo má pokračovat v činnosti.

V metodě **main()** při startu aplikace dochází ke zpracování všech vložených parametrů, na jejichž základě jsou nastaveny vnitřní proměnné, se kterými je dále pracováno. Rovněž je provedeno načtení konfiguračního souboru, které je provedením shodné jako u serverové aplikace. Na závěr metody je volána funkce **mainmethod()**, která již realizuje úkony spojení s vloženými daty. Tato funkce může být volána i vícenásobně v případě použití více vstupních souborů a to použitím cyklu „while“.

V metodě **mainmethod()** je nejdříve obdobným způsobem jako u serveru provedena inicializace knihoven pro práci se sockety. Po ní je provedeno načtení adres z nastaveného souboru a ty jsou ověřeny ve funkci **tokenizer()**, je-li tak zvoleno v konfiguračním souboru. Pokud je využito možnosti ručního vložený adres, je provedeno přímo jejich ověřování na platnost.

Načtení dat ze souboru je provedeno funkcí **read_data()**, která obsahuje vlastní parser na zpracování souboru. Nejprve je provedeno jeho načtení do paměti a následně je zpracován tak, že každý samostatný řádek je brán jako jedna část dat (IP či MAC adresa). Při zpracování dochází v několika cyklech rovněž k odstranění přebytečných odřádkování a mezer. Na závěr jsou veškerá odřádkování nahrazena dělícím znakem parametru. Celý takto načtený a zpracovaný řetězec je poté spolu s operací odesílán k serverové aplikaci. Parser umožňuje i ignorovat určité řádky, například s komentáři a to s využitím znaku „#“ na začátku takového řádku.

V metodě **tokenizer()** je vstupní řetězec rozložen na jednotlivé adresy a ty jsou podle zvolené operace v cyklu ověřovány buď metodou **isip()** v případě použití IP adres či doménových jmen nebo metodou **check_mac()** v případě operace „zapnout“. První jmenovaná metoda ověřuje adresy v serveru rovněž dostupnou metodou

```
ret_msg = getaddrinfo(ip, NULL, NULL, &result);
```

V případě, že je vrácen z této metody chybový kód, je adresa považována za chybnou, v opačném případě je uvolněna paměť proměnné „result“ a metoda vrací návratový kód, značící úspěch. Metoda, ověřující platnost MAC adres je shodná s metodou, jež je využita v serveru.

Po ověření vložených údajů je dále v metodě **mainmethod()** vytvořeno nové vlákno, jehož vstupní funkcí je **vlakno()**. Ta umožňuje i během čekání hlavního vlákna na odpověď od serverové aplikace vypisovat údaje na obrazovku. V tomto případě se jedná o tečky.

Jakmile jsou provedeny všechny tyto úkoly, **mainmethod()** přechází do funkce **communication()**. Ta je rozdělena podmínkou, ověřující použití SSL, na dva různé průchody. V případě využití komunikace bez zabezpečení je nejprve volána funkce **connection()**. Ta se

stará o navázání prvotní komunikace. Adresa i port jsou získány ze souboru s nastavením a jsou použity k tomuto navázání spojení. Klient umožňuje navázat spojení pod protokolem IPv6 i IPV4. Toto nastavení socketu je realizováno API funkcí **getaddrinfo()**. Samotný pokus o připojení probíhá v cyklu

```
for(ptr=result; ptr != NULL; ptr=ptr->ai_next)
```

V tom je využita proměnná „result“, která je získána z **getaddrinfo()**. Obsahuje veškeré nutné informace pro připojení a také určuje protokol, který bude použit. V případě dostupnosti obou komunikačních protokolů, je zvolen nejdříve IPv6 a v případě selhání připojení je při druhém průchodu cyklem zvolen IPv4. Rozdílné nastavení socketu je dosaženo následně

```
soket = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
```

Jakmile je vytvořen socket, je provedeno jeho nastavení do neblokovacího režimu. To je provedeno hlavně z důvodů minimalizace prodlení při připojování, pokud je server odpojen či je krátkodobě nedostupný.

```
ioctlsocket(soket, FIONBIO, &iMode)
```

Hodnota proměnné „iMode“ je rovna jedné, díky čemuž je socket přepnut do v tomto případě použitého neblokovacího režimu. Dále je nastavena časová hodnota, do které má být navázáno spojení a po které je pokus ukončen. V tom případě je cyklus vrácen na začátek nebo ukončen.

```
timeout.tv_sec = 5;
```

Po nastavení čase je proveden již samotný pokus o připojení.

```
i = connect(soket, ptr->ai_addr, (int)ptr->ai_addrlen);
```

Vzhledem k použitému nastavení socketu je následně ověřována řada možných stavů na socketu, které mohou nastat a po kterých je pokus ukončen a cyklus vrácen zpět na začátek. Jedná se především o chyby jako vypršení času a nemožnost se připojit k cílové adrese. Po každé takto zjištěné chybě je socket opět vrácen do blokovacího režimu s využitím **ioctlsocket()** a to změnou hodnoty ve třetím parametru, která je nastavena na hodnotu nula. Pokud je spojení úspěšně navázáno, je socket pouze přepnut zpět do blokovacího režimu a cyklus je ukončen. Přepnutí je provedeno z důvodů snazšího použití socketů, pokud jsou nastaveny jako blokovací. Ještě před samotným ukončením metody je ověřena návratová proměnná z API funkce connect() „i“, zda neobsahuje chybový kód. Tato operace je provedena pro případ, že by veškeré průchody cyklem dopadly neúspěšně a spojení nebylo navázáno.

Po návratu do metody **connection()** je v dalším kroku volána funkce **clear_communication()**. V ní je provedena série kroků, jež je odpovídající svým provedením komunikaci serveru. Jsou při ní využity metody **read()**, **write()** a rovněž jednoduchá interní

šifra. Po získání dat je zavolána metoda **outputs()**, jež obstarává volbu správné metody, která zpracuje již samotný obdržený výsledek. Pomocí podmínek rozhoduje, zda mají být výsledky vypsány na obrazovku, do souboru, případně obojí.

Při výpisu na obrazovku metodou **result_output()** je využito hlavně systémové funkce **strotok()**, jež vložený řetězec rozkládá podle výskytu nastavených dělících znaků. Jednotlivé podřetězce poté vypisuje na obrazovku v cyklu. Rovněž je ověřováno, zda je daný podřetězec adresou, u které bylo požadováno ověření jejího stavu. K tomu je využíváno výše uvedených metod **check_mac()** a také **isip()**. Jakmile je zjištěn výskyt takové adresy, je vypsána vždy na nový řádek a k ní dále po oddělení tabulátorem potřebná informace. Zápis výsledku do souboru probíhá obdobným způsobem, při kterém je použito systémové funkce **fwrite()** pro zápis jednotlivých částí i potřebných oddělovačů jako je odřádkování.

Po dokončení výpisu výsledků je uzavřena samotná komunikace s využitím **communication_close()**, kde jsou v případě nutnosti odstraněny proměnné SSL a především uzavřen samotný socket.

V případě použití zabezpečené komunikace pomocí SSL je postup pro spojení obdobný. Liší se pouze v přidání inicializace SSL struktur jako je „ssl_ctx“ a rovněž použitím odlišných metod pro samotnou komunikaci. V tomto případě se jedná o **ssl_read()** a **ssl_write()**. Ukončení spojení je již shodné jako v případě nezabezpečené komunikace.

Další a již poslední možností je použití operace „status“. V případě jejího zvolení jsou k serveru odeslány 2 požadavky zvlášť. Toho je dosaženo tím, že je nejprve odeslán první požadavek a následně je změněn buffer, ve kterém se tento požadavek nachází na nový metodou **prepare_data()**. V ní je využita funkce **strtok()**, pomocí níž je složen nový požadavek obdobně, jako při jeho tvorbě při načtení. Po dokončení této metody je znovu provedena komunikace se serverovou aplikací a po jejím dokončení je vykonána metoda **prepare_output()**. Ta se stará zpracování obou odpovědí od serveru do jedné tak, aby bylo možné provést výstup na obrazovku či do souboru. V ní je v cyklu postupně rozebírána první odpověď od serverové aplikace a je prováděno hledání shodné adresy v druhé odpovědi s využitím funkce **output_search()**. Adresa stanice i obě odpovědi jsou poté zapsány do nově vytvořené proměnné. Samotné rozdělování odpovědí je realizováno s využitím **getline()**, kde je jako dělicí znak použit „|“.

Po dokončení všech těchto možných průchodů se aplikace navrací do metody **mainmethod()**. V ní je možné nalézt ještě zpracování další možné kombinace parametrů a to s využitím přepínače „c“, jež značí použití cyklu. Tato varianta se od výše popsaných liší pouze tím, že jsou všechny kroky vedoucí ke komunikaci uzavřeny v nekonečném cyklu „while“, který využívá funkce **Sleep()** na pevně stanovenou dobu dvou sekund. Tímto jsou popsány všechny důležité části ovládací konzolové aplikace a jejich fungování.

3.5. Implementace zpracování chyb

Zpracování všech chyb včetně krokových výpisů realizuje metoda **error()** a to jak v serveru, tak i v ovládací aplikaci. Tato metoda je popisována zvlášť především kvůli jejímu hojnému využití napříč oběma aplikacemi. K jejímu volání dochází mnohokrát v téměř každé metodě

aplikace. Přebírá dva parametry, kdy první z nich reprezentuje samotnou chybovou zprávu a druhý kód chyby. Ten může nabývat hodnot 0 až 2 a jejich význam je následující

0 – error

- Značí závažnou chybu, která znemožňuje pokračování v chodu vlákna nebo celé aplikace

1 – warning

- Určuje, že došlo k chybě, která nemá přímý vliv na chod aplikace, ale mohla by způsobit problém

2 – debug

- Vyskytuje se v nejhojnějším počtu a reprezentuje krokové výpisy na obrazovku, které jsou vhodné zejména při ladění či zjišťování problémů

V ovládací aplikaci je v této metodě realizováno pouze vypisování stavů na obrazovku. Obsahuje podmínku, podle které je nejprve určeno, zda program běží v „debug“ režimu, který značí použití krokových výpisů, kdy je na obrazovku vypsáno každé volání této metody. V případě, že není tento režim povolen, probíhá vypisování na obrazovku podle obdrženého kódu chyby. Rovněž může nastat, že bude nastaveno použití režimu bez výstupu na obrazovku. V tom případě tato metoda nebude poskytovat žádné výpisy.

V serveru je tato metoda podstatně obsáhlejší a to především kvůli možnosti využití zápisu chybových zpráv do souboru či systémového logu. Rozhodnutí o způsobu zápisu je řešeno dvojicí podmínek. První z nich řeší, zda je server spuštěn v „debug“ režimu a druhá rozhoduje podle nastavení, zda bude chyba zapsána do systémového logu, souboru či do obou.

V „debug“ režimu jsou povoleny výpisy pouze na obrazovku a server neprovádí při žádné kombinaci nastavení jiný způsob logování. Na obrazovku jsou vypisovány jen zprávy varovné a chybové. Krokové výpisy jsou prováděny až při použití tzv. „verbose“ režimu, jež pracuje společně s „debug“.

Zápis do souboru, je-li povolen, využívá ke své práci knihovnu **log4cplus**. Její prvotní inicializace je popsána v implementaci serverové aplikace. Před samotným zápisem je nutné uzamknout semafor „log_id“. Před samotným zápisem chyby je nutné nastavit podle obdržené chybové proměnné log level

```
myLogger.setLogLevel( ERROR_LOG_LEVEL );
myLogger.setLogLevel( WARN_LOG_LEVEL );
```

Poté je již proveden samotný zápis chyby do nastaveného souboru, po kterém je provedeno již jen odemčení semaforu

```
LOG4CPLUS_ERROR(myLogger, zprava.c_str());
```

Zaznamenání chyby do systémového logu je realizováno rovněž podle aktuálního nastavení serverové aplikace. Samotná zapsaná informace je pod všemi platformami shodná, ovšem rozdíl je v implementaci.

Windows:

Pod tímto systémem je pro zapsání do systémového logu využito systémových API funkcí. Základní inicializace proměnných a otevření logu bylo popsáno již v implementaci samotného serveru. V samotné metodě **error()** je provedeno jen několik kroků, nutných pro vykonání zápisu. Prvně je zpracován řetězec se vstupní zprávou. API funkce využívá řetězec typu LPCSTR neboli „long pointer to constant string“. K jejímu použití je nejprve nutné provést převod řetězce se vstupní zprávou z typu „string“, který je využit z důvodu kompatibility s verzí serveru pro systém Linux.

```
CStringA temp(zprava.c_str());
LPCSTR lpcstr = temp;
```

Tento převod je poměrně komplikovaný, ovšem bez jeho provedení by bylo nutné provedení komplexní úpravy serveru pod platformou Windows a to by nebylo vhodné především z důvodů kompatibility. Pro samotný zápis takto převedeného řetězce do systémového logu je použita funkce

```
ReportEventA(errorHandle,
             EVENTLOG_ERROR_TYPE, // typ události
             0, // žádná kategorie
             11,
             NULL, // žádný uživatelský bezpečností ukazatel
             1,
             0, // žádná data
             &lpcstr, // ukazatel na řetězec s chybou
             NULL)
```

Prvním jejím parametrem je proměnná logu, vytvořená při inicializace aplikace, následovaná typem zprávy. Mezi nevyužité parametry, kterým je ponechána výchozí hodnota, patří především nastavení kategorie, identifikátor události, bezpečností ukazatel a také přiložená data. Předposledním použitým parametrem je ukazatel na převedený řetězec s chybovou zprávou.

Jakmile je zpráva úspěšně uložena v logu, je tato sekce metody opuštěna. Mimo použití typu události „EVENTLOG_ERROR_TYPE“ pro chybu je využit v alternativním případě „EVENTLOG_WARNING_TYPE“ pro varovné zprávy.

Linux:

V případě tohoto operačního systémů je zápis do systémového logu jednodušší a to kvůli možnosti využít pro zápis datový typ „char*“, který lze snadno získat z používaného „string“. Pro zápis jsou použity 2 funkce

```
openlog("Bcon server", NULL, LOG_DAEMON);
syslog(LOG_DAEMON | LOG_ERR, zprava.c_str());
```

První z těchto funkcí obstarává otevření příslušného systémového logu, kterým je v tomto případě log pro zápis zpráv od démonů. Rovněž je uveden jako první parametr název, který bude zobrazován jako aplikace, jež provedla zápis. Po otevření je dokončeno uložení chybové zprávy. Při těchto operacích je rovněž využit příslušný semafor pro „log_id“, jelikož v používané verzi systému Ubuntu nebyly ještě implementovány ekvivalentní funkce **openlog_r()** a **syslog_r()**, které je možné využít i bez použití semaforů.

Tímto je popsán způsob zpracování chybových a informačních zpráv serverovou i ovládací aplikací. Použitou metodu lze snadno rozšířit o další funkcionalitu a taktéž úrovně logování.

4. Nasazení systému

Nasazení systému na cílové platformy tvoří nedílnou součást při jeho tvorbě. Při této činnosti je nutné zajistit správné umístění všech souborů, registraci služeb, nastavení spouštění aplikace a řadu dalších úkolů. Rozdíly při instalaci serverové i klientské aplikace mezi systémy Linux a Windows jsou propastné z důvodu jejich odlišné koncepce. Proto bude každá platforma rozebrána zvlášť a bude popsáno jako správné použití prostředků pro instalaci, tak i jejich rozmístění.

4.1. Linux

Instalace démona na platformě Linux a v tomto případě použitém systému Ubuntu je obdobná jako u ostatních aplikací pro tuto platformu. Ubuntu a veškeré ostatní distribuce odvozené od Debianu využívají při instalaci software balíčkovací systém, jehož soubory mají příponu „.deb“ a je možné tak instalovat jak oficiální knihovny z repositářů, tak i ručně vlastní knihovny.

Nejdříve je při tvorbě vlastního balíčků nutné vytvořit adresářovou strukturu shodnou s cestami, kam budou při samotné instalaci umístěny potřebné soubory. V případě serveru se jedná o cesty

/opt/bcontrols/

- Obsahuje samotný spouštěcí soubor aplikace **bcontrols** a soubor pro obnovení konfigurace **restore**

/opt/bcontrols/cert

- Jsou v něm umístěny základní certifikáty **server.key** a **server.pem** použité při zabezpečené komunikaci a rovněž soubor certifikační autory **authority.pem**

/etc/

- Tato cesta obsahuje konfigurace všech nainstalovaných aplikací, čili v tomto případě jsou do ní umístěny soubory s nastavením **bcontrols.conf** a **bhosts.conf**, ve kterém je případný seznam adres s povoleným přístupem k serveru

/etc/init.d/

- Nachází se zde soubor **bcontrols**, zajišťující spouštění a zastavení chodu démona. To je realizováno s využitím příkazu „start-stop-daemon“ a příslušnými parametry

Rovněž je také nutné vytvořit část adresářové struktury, která je využita pouze při samotné tvorbě balíčku a obsahuje údaje jako kontrolní součet a úkony, které mají při instalaci a případném odinstalování vykonány

/DEBIAN/

Obsažené soubory:

conffiles

- V tomto souboru jsou označeny názvy souborů, které obsahují konfiguraci a nemají být odstraněny při odinstalování aplikace. Rovněž zajišťuje, že nebudou přepsány při instalaci nové verze programu.

control

- Nalézá se zde kompletní popis a informace o instalované aplikaci. Mezi ně patří verze, obsažená v balíčku, velikost nainstalované aplikace, cílová platforma a rovněž důležitý údaj v podobě závislostí na dalších balíčcích. Ty se díky tomuto údaji automaticky doinstalují do systému před samotnou instalací aplikace.

md5sums

- Jsou zde umístěné kontrolní součty všech souborů, které jsou popsány v první části balíčku a budou instalovány na koncový systém. Zajišťují, že balíček nebude nainstalován v případě, že bude poškozen a nebudou tak odpovídat kontrolní součty.

postinst

- Obsah tohoto skriptu obsahuje kroky, které mají být vykonány po dokončení instalace aplikace. V tomto případě jsou prováděny tyto úkoly
    - **update-rc.d bcontrols start 51 S .** – provádí registraci nainstalovaného démona do systému, aby tak došlo k jeho spuštění se startem operačního systému
    - **chmod +x /etc/init.d/bcontrols –** přidává právo na spuštění je spouštěcímu skriptu démona
    - **/etc/init.d/bcontrols start –** provádí samotné spouštění démona s využitím start-stop-daemon, jež je umístěno uvnitř tohoto souboru. Tento příkaz s různými parametry rovněž zajišťuje jeho ukončení či restartování

postrm

- Skript obsahuje jedinou operaci, která je vykonána po odebrání aplikace ze systému
    - **update-rc.d -f bcontrols remove –** obstarává odebrání záznamu pro start démona s operačním systémem

prerm

- Stará se o provedení kroků před samotným odebráním programu z OS. Nalézá se zde pouze jeden krok
    - **/etc/init.d/bcontrols stop** – provede zastavení démona, aby mohl být odebrán. Využívá start-stop-daemon

Jakmile je úspěšně vytvořena adresářová struktura a umístěny veškeré soubory, které budou instalovány, je nutné pro ně provést výpočet kontrolních součtů, které budou umístěny ve výše popsaném souboru **md5sums**. K tomu slouží tento příkaz

```
find * -type f ! -regex '^DEBIAN/.*' -exec md5sum {} \; > DEBIAN/md5sums
```

Jeho spuštění je potřeba provést v příkazovém řádku, nastaveném na adresář, ve kterém se nalézají jednotlivé kořenové adresáře instalace, čili **DEBIAN**, **etc** a **opt**. Jakmile jsou kontrolní součty vypočteny, je možné přejít k tvorbě samotného balíčku. K tomu je využit níže uvedený příkaz, který by měl být prováděn pod účtem **root**, jelikož by bez něj mohlo dojít k vytvoření bezpečností díry. Pokud by bylo provedeno sestavení pod současným uživatelem, byl by jejich vlastníkem na cílovém PC uživatel se stejným UID.

```
dpkg-deb -b install bcontrols-0.033-i386.deb
```

Výsledkem příkazu je balík, jehož název je udán jako poslední parametr. Balík je vytvářen z adresáře „install“, ve kterém se nachází výše popsaná balíčková struktura. Nyní je možné jej přenášet a instalovat na všechny operační systémy, jež využívají shodný balíčkovací systém a architekturu. V tomto případě **i386**. Serverová i konzolová ovládací aplikace jsou vytvořeny rovněž pro cílovou platformu **amd64**. Rozdíl lze rozpoznat snadno a to především díky uvádění platformy do názvu balíčku.

V případě instalace konzolové aplikace pro ovládání serveru je tvorba balíčku prováděna obdobným způsobem jako v případě démona. Rozdíly se nacházejí pouze v nepřítomnosti souborů jako **prerm**, **postrm** a **postinst**, které nejsou potřebné a také v samotných instalovaných souborech.

4.2. Windows

Instalace služby na platformě Windows je velmi podobná instalacím ostatních aplikací na této platformě, stejně jako v případě instalací na platformě Linux. Značným způsobem se však liší způsoby tvorby instalátorů, které se vytvářejí s využitím různých dostupných aplikací k tomu určených. Mezi nimi jsou mnohdy velké rozdíly ve způsobu a komplexnosti. Nejjednodušším způsobem tvorby je využití samo-rozbalovacích instalačních archivů, které je možné vytvořit aplikacemi jako je například **Winrar**. Na druhé straně lze využít prostředků podstatně komplexnějších, mezi které patří i **NSIS**, který byl k tvorbě instalátoru zvolen.

Jedná se Open Source systém, který umožňuje do značné míry s využitím skriptovacího jazyka upravovat a kontrolovat jednotlivé části instalace. Rovněž je do něj možné doplňovat další funkčnost na základě pluginů, kterých je využito v případě práce se službou. Hlavním prvkem je soubor s příponou **.nsh**, ve kterém jsou veškeré příkazy k vytvoření instalátoru. Tento zdrojový soubor je umístěn v adresáři, kde jsou umístěny všechny soubory, které budou při instalaci použity. V tomto případě není nutné vytvářet žádnou adresářovou strukturu jako v případě systému Linux s balíčkovacím systémem, postaveném na **.deb**. Vytvoření samotného instalátoru je provedeno spuštěním kompilátoru NSIS na uvedený soubor se skripty. To lze provést v případě správně nainstalovaného kompilátoru pouhým

dvojitým klikem na samotný skript. V kompilátoru jsou buď vypsány zjištěné chyby, nebo je vytvořen instalační soubor, jehož název je nastaven ve skriptu.

4.2.1. NSIS skript

První krokem v nově vytvořeném skriptu je vložení případných dalších použitých skriptů či pluginů, které budou využity s pomocí **include**

!include "MUI2.nsh"

Tímto je vloženo použití knihovny, reprezentující moderní UI. Lze použít různé druhy vzhledů samotného instalátoru právě tímto způsobem. Pluginy, jež jsou umístěny ve složce k tomu určené u samotné NSIS aplikace, není nutné přímo vkládat s využitím **include**. Dále je ve skriptu možné provést nastavení důležitých údajů, které budou používány v dalších jeho částech

Name "Binarry Control Server"

- Název, který bude zobrazen v hlavičce instalátoru

OutFile "BcontrolsInstall.exe"

- Výstupní název vytvořeného instalátoru

RequestExecutionLevel Admin

- Instalátor bude při spuštění vyžadovat administrátorská práva

InstallDir "$PROGRAMFILES\Bcontrols"

- Zvolíme základní instalační cestu

Rovněž můžeme nastavit interface, aby zobrazoval případné chyby při instalaci a také nastavit makro na jazyk, který má být použit instalátorem

!define MUI_ABORTWARNING
!insertmacro MUI_LANGUAGE "English"

Důležitým krokem je nastavení stránek, které mají být při instalaci zobrazeny. Jedná se například o stránku s licenčním ujednáním, stránku s volbou instalační cesty a další. Tvůrce skriptu si takto může jednoduše zvolit, co se má uživateli zobrazit a co může tímto způsobem upravovat. Příkladem stránky může být právě nastavení instalační cesty

!insertmacro MUI_PAGE_DIRECTORY

Různé části instalace jsou ve skriptu umístěny do sekcí, odlišených použitými názvy. Lze vytvářet i skryté sekce, které díky tomu nebudou viditelné pro uživatele. Toho je v případě server využito u sekce, která se stará odebrání předchozí verze aplikace.

V samotné instalaci probíhá nastavení cest a vytvoření potřebných adresářů následujícím způsobem. Je při něm využito i systémových proměnných, což je vhodné zejména při umisťování konfiguračních souborů a uživatelských dat. V tomto případě se jedná o proměnnou **ALLUSERSPROFILE**, která určuje použití adresáře „ApplicationData“ či „AllUsers“ a to z důvodů instalace služby, která musí mít přístup ke svým datům i v případě, že není přihlášený žádný uživatel. Použité soubory musí být umístěny v adresáři se skriptem.

```
SetOutPath "$%ALLUSERSPROFILE%\Bcontrols"
        File "bcontrols.conf"
CreateDirectory "$%ALLUSERSPROFILE%\Bcontrols"
```

V závěru instalačního skriptu je využito rovněž možnosti spuštění další aplikace pomocí příkazu **ExecWait**. Ten zajistí, že instalátor počká na dokončení takto spuštěné aplikace či dalšího instalátoru. Je použit k instalaci „Visual C++ Redistributable“ knihoven a také „.NET Framework 2.0.“. Po dokončení jejich instalace jsou odstraněny jejich instalátory s využitím příkazu **Delete**.

Jakmile jsou dokončeny všechny tyto úkony, je na úplný závěr provedena instalace samotné služby do systému a její spuštění. K tomu je využit plugin **Simple Service**, jež je dostupný na stránkách tvůrců NSIS a rovněž se na něj nalézá odkaz ve zdrojích [6]. Nejprve je do systému nainstalována služba

```
SimpleSC::InstallService "Bcontrols" "Binnary Control Server" "16" "2" "$INSTDIR\bcontrols.exe" "" "" ""
```

Tento příkaz obsahuje řadu parametrů, využity jsou ovšem jen některé, například nastavení názvu, jež bude uveden ve správci služeb, dále závislosti na jiných službách či nastavení, že služba bude spouště pod systémovým účtem. Samotné její spuštění je prováděno jednoduchým příkazem

```
SimpleSC::StartService "Bcontrols" ""
```

Při dokončení instalace je nutné vytvořit i skript pro odebrání aplikace. K tomu je využita sekce, která je rovněž umístěna ve skriptu. Samotný soubor pro odinstalaci se vytváří takto

```
WriteUninstaller "$INSTDIR\BcontrolsRemove.exe"
```

V něm obsažená sekce obstará nejprve zastavení a odstranění služby ze seznamu nainstalovaných služeb. Při použití příkazu pro zastavení je použit parametr „1“. Ten značí, že skript počká na ukončení služby, než bude pokračovat.

```
SimpleSC::StopService "Bcontrols" 1
SimpleSC::RemoveService "Bcontrols"
```

Po jejím odebrání je provedeno již jen odstranění všech souborů a adresářů. To je provedeno s využitím příkazu **Delete** a také **RMDir** pro adresáře. Nejsou odstraněny pouze soubory s nastavením, které mohou být využity při pozdější opětovné instalaci aplikace.

Pro tvorbu instalačního souboru konzolové ovládací aplikace byl využit velmi obdobně vytvořený skript s drobnými rozdíly. Hlavním je nevyužití pluginu pro práci se službami.

5. Závěr

Doplnění nově požadované funkčnosti do systému Netcon se v průběhu tvorby této práce ukázalo jako poměrně náročný úkol, který si vyžádal v podstatě kompletní přepracování jeho návrhu. Původní zůstaly pouze některé části, jako je například rozvržení tříd. Díky tomuto přepracování se však stal systém podstatně rychlejší a rovněž spolehlivější a bylo možné jej lepé testovat a ladit.

U serverové aplikace se ukázala jako nejkomplikovanější implementace podpory pro protokol IPv6, která zabrala spolu s tvorbou paměťové databáze pravděpodobně nejvíce času. Navíc pro implementaci tohoto protokolu není na internetu dostupné ještě takové množství zdrojů jako v případě protokolu IPv4, ze kterých je možné čerpat. Právě díky omezenému množství zdrojů jsou funkce jako je WOL či ping pod tímto protokolem implementovány pouze v základní verzi a je možné u nich funkčnost ještě dále rozšiřovat. Samotný převod serveru pod platformu Windows se obešel bez vážnějších problémů v kompatibilitě zdrojových kódů či při tvorbě knihoven, využitých pod touto platformou. Díky obsaženému programovému základu pro tvorbu služby ve Visual Studiu rovněž nebyly vážnější problémy při její tvorbě. Jako nesložitější se ukázalo otestování inicializace nainstalované služby, u které se velmi obtížně provádí ladění.

Nejpodstatnějších změn dosáhla konzolová ovládací aplikace, která je vytvořena s využitím několika málo částí předchozí verze. Zásadní rozdíl nastal v použití tříd, jelikož původní aplikace je žádným způsobem nevyužívala. Během jejího vývoje se nevyskytly žádné vážnější problémy.

Takto vytvořený systém byl testován na více než 100 počítačích a bylo tak ověřeno jeho možné praktické využití při kontrole většího množství stanic naráz. Důvod poměrně malého počtu komplikací při vývoji byl pravděpodobně zapříčiněn především již dřívější zkušeností s vývojem bakalářské práce „Systém pro vzdálené zapínání PC na učebnách“, která využívala podobných prostředků pro tvorbu.

6. Literatura

[1] PRATA, S. *Mistrovství v C++*. Praha: Computer Press, 2007. 957 s. ISBN 80-7226-339-0

[2] SATRAPA, P. *Internetový protokol IPv6*. Praha: CZ.NIC, 2008. 351 s. ISBN 978-80-904248-0-7

[3] Computer Education - Environment Variables in Windows XP
http://vlaurie.com/computers2/Articles/environment.htm

[4] Stuart's Useful C/C++ Pages
http://www.jb.man.ac.uk/~slowe/cpp/itoa.html

[5] Gettimeofday functionality
http://social.msdn.microsoft.com/Forums/en/vcgeneral/thread/430449b3-f6dd-4e18-84de-eebd26a8d668

[6] nullsoft scriptable install system - NSIS Simple Service Plugin
http://nsis.sourceforge.net/NSIS_Simple_Service_Plugin

[7] nullsoft scriptable install system
http://nsis.sourceforge.net/Main_Page

[8] POSIX thread (pthread) libraries - tutorial
http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

[9] MSDN Library - _beginthread, _beginthreadex
http://msdn.microsoft.com/en-us/library/kdzttdcb.aspx

[10] libConfuse library
http://www.nongnu.org/confuse/

[11] Jak na OpenSSL; Michal Kára
http://www.root.cz/clanky/jak-na-openssl/

[12] log4cplus project
http://log4cplus.sourceforge.net/

[13] logging made easy in your c++ applications; Ramanjaneyulu Malisetti
http://www.codeproject.com/KB/cpp/Log4cplus.aspx

[14] OpenSSL Tutorial - Client
https://thunked.org/programming/openssl-tutorial-client-t11.html

[15] MSDN Library – ControlService function
http://msdn.microsoft.com/en-us/library/ms682108(VS.85).aspx

[16] Open Group – getnameinfo function
http://pubs.opengroup.org/onlinepubs/009695399/functions/getnameinfo.html

[17] Ubuntu wiki - Vytvoření .deb balíku
http://wiki.ubuntu.cz/Vytvoření%20.deb%20balíku